

UNIwersytet Jagielloński
Wydział Fizyki, Astronomii
i Informatyki Stosowanej

Tworzenie narracji komputerowej gry fabularnej z użyciem transformacji grafowych

Autor: mgr Iwona Grabska-Gradzińska

Promotor: prof. dr hab. Ewa Grabska

Promotor pomocniczy: dr Wojciech Palacz

Kraków 2022

Składam gorące podziękowania Pani Profesor Ewie Grabskiej,
która była dla mnie, *nomen omen*, jak matka.

Dziękuję także dr Wojciechowi Palaczowi za niestrudzoną
pomoc i wsparcie.

Spis treści

Wstęp.....	7
1 Narracja w grach na tle innych utworów fabularnych.....	11
1.1 Struktury grafowe w odniesieniu do konstrukcji tekstu literackiego	11
1.2 Narracje nieliniowe.....	13
1.3 Typologia gier komputerowych w kontekście modelu grafowego	15
1.4 Struktury narracyjne w teorii literatury	21
2 Case study	26
2.1 Fabuła gry przykładowej	26
3 Model grafowy.....	29
3.1 Formalne definicje grafowe.....	33
4 Założenia systemu StoryGraph.....	49
4.1 Struktura grafu wielospokowego.....	51
4.2 Referencje do wierzchołków	53
4.3 Struktura predykatów stosowalności	55

4.4	Struktura instrukcji	55
4.5	Format zapisu danych.....	59
4.6	Ograniczenia przyjętych rozwiązań.....	64
5	Implementacja.....	69
5.1	Algorytmy stworzone na potrzeby implementacji.....	70
5.2	Dane wejściowe.....	78
5.3	Architektura implementacji referencyjnej	78
5.4	Funkcjonalność implementacji referencyjnej	83
6	Zastosowania systemu	97
6.1	Model grafowy a mechanika gry	97
6.2	Proces projektowy	114
6.3	Gry oparte o system StoryGraph	129
	Podsumowanie	133
	Bibliografia.....	136
	Spis gier wymienionych w pracy	149
	Spis ilustracji	151
	Dodatek A – Objaśnienia symboli użytych w definicjach	153
	Dodatek B – Słowniczek pojęć	155
	Dodatek C – JSON Schema	158
	Dodatek D – Wybrane misje gry fabularnej stworzone przez studentów	165

Wstęp

Wśród wszystkich systemów interaktywnych o ustalonych zasadach nazywanych grami wyróżniamy kategorię gier komputerowych, a w niej, spośród wszystkich gatunków, wyodrębnijmy gatunek fabularnych gier przygodowych.

Fabularne przygodowe gry komputerowe są dziedziną twórczości znajdującą się pomiędzy literaturą a informatyką. Od strony literaturoznawczej są przedmiotem zainteresowania przede wszystkim narratologów, czyli teoretyków literatury zaangażowanych w badania nad fabułami oparte o metody strukturalne i zajmujące się warstwą narracyjną utworu (dowolnego, badania nie ograniczają się do literatury pięknej, czy nawet słowa pisanego) w kontekście jego głębokiej struktury formalnej.

Z jednej strony są więc gry fabularne tekstem, utworem literackim w najszerszym rozumieniu narratologicznym¹. Jest to dzieło specyficzne, mające co najmniej dwóch autorów: twórcę gry i gracza, a żywotność dzieła ograniczona jest czasowo do końca rozgrywki. Z drugiej strony gra jako gatunek rządzi się swoimi prawami, a wpływ technologii na ostateczny kształt dzieła jest większy niż w jakiegokolwiek innej dziedzinie fabularnej, nawet w filmie. Związane jest to wbrew pozorom nie z problemami renderowania grafik i wydajnością kart graficznych, bo to nie warstwa wizualna stanowi o specyfice gier wideo, mimo że sama nazwa gatunku sugeruje prymat wizualności.

¹ Nie będziemy się jednak w niniejszej pracy posuwać tak daleko jak Roland Barthes, jeden ze znanych narratologów, w swoim słynnym stwierdzeniu, że „wszystko jest tekstem”.

Podstawową cechą dystynktywną gier w stosunku do innych dziedzin sztuk fabularnych jest nieliniowość narracji. Twórca gry – a w zasadzie pierwotny twórca gry – określa tylko ramy fabularne utworu: stan początkowy świata, cel do osiągnięcia i mechanikę gry oraz projektuje pewne punkty węzłowe akcji. Końcowy kształt pojedynczej fabuły jest dziełem ostatecznego twórcy – czyli gracza – i efektem decyzji podejmowanych przez niego w trakcie rozgrywki. Oczywiście jest możliwe takie zaprojektowanie gry, by była ona *de facto* fabułą liniową, a jedyna niepewność dotyczyła tego, czy osiągniemy cel, czy nie. Możliwe jest też dawanie graczowi wyborów pozornych, kiedy niezależnie od dokonanych wyborów dochodzi do tego samego punktu węzłowego. Im bardziej otwarty jest świat, tym większa różnorodność potencjalnych możliwych fabuł.

Dwuetapowość procesu kreacji, mnogość wariantów powoduje, że formalny opis fabuł gier tworzony czy to dla potrzeb porównawczych, czy do celów sprawdzania spójności ram narracyjnych, czy jako podstawa do automatycznego lub półautomatycznego generowania fabuł – jest dużym wyzwaniem. Nie może to być formalizm oparty na teoriach czysto literaturoznawczych, choć oczywiście będzie bazował na osiągnięciach w dziedzinie formalizacji struktur narracyjnych, dokonanych przede wszystkim przez badaczy z nurtu strukturalizmu w pierwszej połowie XX wieku. Z punktu widzenia najnowszych osiągnięć teorii literatury ten typ opisu procesu twórczego jest już dość archaiczny, wyparły go teorie nowoczesne i postnowoczesne, ale na styku literatury i informatyki nadal sprawdza się bardzo dobrze.

Szukając inspiracji teoretycznoliterackich warto sięgnąć po teorie kładące nacisk na wariantowość fabuł. W niniejszej pracy, w ramach poszukiwań literaturoznawczych, sięgnięto po teorię Władimira Proppa, rosyjskiego formalisty, którego teoria morfologii bajki magicznej zainspirowała nie tylko autorkę niniejszej rozprawy, ale także wielu innych badaczy struktur literackich na gruncie informatyki. Z kolei na gruncie algorytmicznym najlepiej nadające się do oddania zależności narracyjnych fabuł okazały się transformacje grafowe z procedurą generacyjną przydatne do specyfikacji, modelowania i prototypowania narzędzi projektowych opartych na wiedzy. Narzędzia oparte na transformacjach pozwalają opisać nie tylko aktualny stan systemu, ale także jego dynamiczne zmiany w czasie.

Połączenie tych dwóch modeli teoretycznych zaowocowało modelem formalnym przygodowych gier komputerowych, zaprezentowanym czytelnikowi niniejszej pracy. Model ten stara się odwzorować schematy myślowe twórców gier i zależności między elementami świata gry po to, aby system uprościć tam, gdzie uproszczenia formalne zgodne są z ograniczeniami narzucanymi przez struktury narracyjne i dzięki temu udostępnić narzędzia do skutecznego prze-

tworzenia świata gry zgodnie z decyzjami podejmowanymi przez gracza. Może być wykorzystywany do porównawczej analizy rozgrywek jednej gry, gier między sobą, do wspomagania projektowania struktur narracyjnych gry a nawet być podstawą do stworzenia systemu automatycznego generowania rozgrywki.

*

Praca składa się z sześciu rozdziałów i czterech dodatków.

W **rozdziale pierwszym** zaprezentowano kategorię gier przygodowych jako formę gry komputerowej na tle typologii gatunku oraz przedstawiono literaturoznawcze fundamenta teoretyczne, na których opiera się niniejsza praca. Wprowadzono też czytelnika w zagadnienia roli modeli formalnych we wspomaganiu projektowania.

Rozdział drugi przedstawia praktyczny przykład gry, który będzie ilustracją tez zawartych w pracy i generatorem schematów fabularnych w niej omawianych. Gra ta była podstawą twórczej pracy projektowej grup studentów w latach 2019–2022 i obserwacja tej pracy była cennym elementem weryfikacji stawianych tez i kopalnią materiałów problemowych. Studenci okazali się też niestrudżonymi testerami rozwiązań implementacyjnych zaproponowanych w pracy.

Rozdział trzeci zawiera formalne definicje matematyczne systemu grafowego będącego teoretyczną podstawą systemu. Wprowadzone jest tam pojęcie grafu n -warstwowego, grafu snopkowego (struktury grafowej stworzonej na potrzeby modelowania gier przygodowych) oraz dopasowania generycznego jako narzędzia określającego możliwy zakres modyfikacji i produkcji generycznej jako bezpośredniego odwzorowania akcji gracza.

Rozdział czwarty i piąty jest przedstawieniem specyfikacji systemu informatycznego służącego do praktycznego wykorzystania zdefiniowanego modelu teoretycznego oraz analiza implementacyjna modelu, z uwzględnieniem opisu algorytmu dopasowania lewej strony produkcji do świata gry i możliwości jej aplikacji. Implementacja referencyjna przywoływana w pracy znajduje się w repozytorium <https://github.com/iwonagg/StoryGraphPhD>.

Rozdział szósty przedstawia możliwe zastosowania systemu opartego o model grafowy w wielu aspektach: jego znaczenia dla tworzenia mechaniki gry, opartym na nim procesie projektowym, ze szczególnym uwzględnieniem projektowania zespołowego (ang. *collaborative design*), wykorzystywania mechanizmów grafowych do tworzenia gry w modelu bardziej lub

mniej otwartego świata, analizy zachowań gracza a także oceny wpływu gry na schematy poznawcze gracza w grach poważnych (ang. *serious games*). Zasygnalizowana jest też możliwość wykorzystania mechanizmów systemu do analizy grywalności.

Dodatki do pracy obejmują objaśnienia symboli użytych w formalizmach matematycznych, słowniczek pojęć literaturoznawczych i groznawczych, definicję struktury schematu JSON Schema umożliwiającą walidację poprawności danych wytworzonych w trakcie prac projektowych oraz diagramy projektowe wybranych misji fabularnych stworzonych przez studentów.

Tezy pracy

Celem rozprawy jest opracowanie koncepcji opisu schematów fabularnych komputerowych gier przygodowych w oparciu o reprezentację grafową i grafową procedurę generacyjną. Przedstawione rozwiązanie w rozprawie zostało zweryfikowane praktycznie przez zaimplementowanie mechanizmów symulatora rozgrywki gracza gier komputerowych.

Zaproponowany w rozprawie nowy model grafowy wraz procedurą generacyjną jest odpowiedni do:

- opisu struktur fabularnych przygodowych gier wideo,
- implementacji mechanizmów symulatora rozgrywki gracza fabularnych gier przygodowych w celu testowania linii fabularnych,
- tworzenia mechanizmów współpracy projektowej w zespołowym tworzeniu gier,
- analizowania fabuł formalnie zakodowanych w postaci grafu rozgrywki, zarówno przeprowadzonych przez gracza w czasie rzeczywistym jak i hipotetycznych.

1 Narracja w grach na tle innych utworów fabularnych

Pomysł uporządkowania narracji² na bazie modelu grafowego wynika w naturalny sposób z możliwości struktury grafowej w modelowaniu związków przyczynowo-skutkowych, a z drugiej strony dążenia do uporządkowania struktur narracyjnych uwzględniającego możliwości i ograniczenia techniczne nośnika narracji. Nośnikiem w tym najbardziej tradycyjnym sensie jest tekst na łamach tradycyjnej książki i większość modeli badawczych dotyczących narracji wywodzi się ze studiów nad literaturą piękną. Możliwości technologiczne nie wniosłyby do rozwoju teorii literackich zbyt wiele, gdyby nie umożliwiały włączenia odbiorcy w proces tworzenia dzieła. Jest to ogromne wyzwanie dla autora, którego nazwiemy w tym momencie już bardziej projektantem fabuły, aby każda wybrana przez odbiorcę ścieżka narracyjna była wewnętrznie spójna i nie łamała zasad świata przedstawionego.

1.1 Struktury grafowe w odniesieniu do konstrukcji tekstu literackiego

Zanim przejdziemy do definiowania grafowego modelu gier fabularnych, zastanówmy się, jakie porządki elementów fabuły możemy zauważyć w dziele literackim, począwszy od tych najbardziej tradycyjnych opowieści przekazywanych ustnie lub zapisanych na archaicznych

² Wybrane pojęcia związane z teorią literatury i badaniami gier opisane są w słowniczku na s. 149.

nośnikach tekstu – zwojach czy kartach kodeksów. Zauważymy, że rozwój możliwości technologicznych zawsze znajdował odwzorowanie w rozbudowie struktur fabularnych i zmianach sposobu korzystania przez użytkownika z tekstu, choć niekiedy powodował nie tyle powstanie, co rozpowszechnianie się pomysłów fabularnych, które bez pomocy technologii były trudne do osiągnięcia.

Pierwszym możliwym uporządkowaniem struktury narracyjnej był porządek liniowy, wymuszany przez formę zwoju, a wcześniej narzucany przez konwencję opowieści oralnej, gdzie należy podążać za tokiem mowy autora bez możliwości opuszczenia fragmentu, nie mówiąc o zmianie kolejności elementów opowieści. Wszelkie zaburzenia liniowości narracji musiały być tam elementem kompozycji utworu, a nie sposobem odbioru. Sama kompozycja takiego utworu nie jest ograniczona niczym poza pamięcią autora, ale też właśnie mnemotechniczne sposoby porządkowania opowieści bardzo silnie odciskają się na jej formie. Budowa opiera się na strukturze punktów węzłowych, wypełnionych treścią improwizowaną, zmieniającą się wraz z okolicznościami, audytorium, czy też po prostu nastrojem narratora. Uporządkowanie kompozycji tekstu ma charakter przyczynowo-skutkowy. Przestrzeń pomiędzy punktami węzłowymi może podlegać dynamicznej modyfikacji, warianty tekstu są bardzo ulotne, funkcjonują tyle, ile trwają w pamięci słuchaczy. Im większy nacisk kładzie autor na niezmiennosc i powtarzalność tekstu, tym bardziej wyrafinowana jest struktura formalna utworu – pojawiają się rymy, segmentacja utworu (strofy, struktury dialogiczne, figury retoryczne, paralelizmy, czy rozmaite cechy gatunku i konwencje ułatwiające odtworzenie historii). Segmentacja i struktura utworu służą bardziej autorowi do łatwiejszego zapamiętania i odtworzenia go, niż słuchaczowi.

W porządku drzewowym, opartym na spisie treści, występującym najczęściej w tekstach niebeletrystycznych, czytając tekst wychodzimy od początku i wybieramy interesującą nas ścieżkę lub kilka ścieżek lektury idąc coraz bardziej w głąb drzewa spisu treści. Zwróćmy uwagę, że wybór jednej ścieżki nie unieważnia pozostałych: ścieżki się uzupełniają. Ścieżki lektury mają określony kierunek, ponieważ po wyodrębnieniu gałęzi fragmenty składające się na nią czytamy już w sposób liniowy.

Sieć od drzewa różni się tym, że dany fragment może mieć wiele poprzedników i wiele następników w ścieżkach lektury. Wyzwaniem technologicznym jest umożliwienie czytelnikowi wybrania ścieżki w sposób łatwy i nierozbijający doświadczenia lektury oraz ewentualne uzyskanie równorzędności poszczególnych sekwencji. W grach jest to sztandarowy sposób obcowania z dziełem, ale wbrew pozorom nie jest najczęstszą formą narracji.

Współcześnie strategia odbiorcy zależy od celu, w którym sięga po lekturę, ale porządek liniowy jest w dalszym ciągu najbardziej popularnym i powszechnie uważanym za podstawowy sposobem obcowania z tekstem beletrystycznym a wbrew pozorom jest także podstawą odbioru w większości tekstów kultury wymagających współtworzenia przez użytkownika, także w grach wideo. Modelowy odbiorca czyta utwór od początku do końca, „z lewa w prawo” i to ten model jest punktem odniesienia dla autora konstruującego utwór. Mamy techniczną możliwość wyboru bardziej oryginalnej ścieżki lektury, nadal jednak w przypadku tradycyjnej lektury pamiętamy o prymarnym liniowym porządku i jeśli go modyfikujemy, to na własną odpowiedzialność, choć oczywiście statystycznie częściej, jeśli jesteśmy do tego zachęceni widoczną hierarchią treści.

1.2 Narracje nieliniowe

Istotnym *novum* w recepcji struktury narracyjnej stało się dopiero wprowadzenie realnej decyzyjności czytelnika w stosunku do sekwencji zdarzeń, którym podlegają bohaterowie. Dzięki oddaniu odbiorcy inwencji w tworzeniu narracji mamy pierwszy w historii narratologii przykład splatających się narracji nieliniowych. Poszczególnych narracji w takim utworze jest znacznie więcej, niż mógłby nawet wyobrazić sobie autor klasycznej hipertekstowej powieści, gdzie każda ścieżka lektury musiała być liniowo zaprojektowana przez autora i wpleciona w inne wątki. Tutaj mamy do czynienia z regułami opisującymi świat i warunki, w ramach których użytkownik generuje swoją własną narrację. Z tego względu formalny opis węzłów narracyjnych musi być poprzedzony przez formalny opis świata i reguł nim rządzących. W odróżnieniu od klasycznej opowieści tekstowej bowiem autor nie tyle szuka sekwencji akcji, które mieszczą się w przyjętym przez niego opisie świata, ale takich reguł rządzących światem, aby każda sekwencja akcji wygenerowana w oparciu o reguły mieściła się w ramach zdefiniowanego świata.

Wbrew pozorom oddanie czytelnikowi możliwości współtworzenia ścieżki narracji nie jest domeną li tylko gier wideo. Eksperymenty formalne pozwalające na kilka sposobów lektury pojawiały się w historii dużo wcześniej niż wynaleziono prąd, nie mówiąc o Internecie. Funkcjonowały jednak na prawach kuriozum, a nie jako typowa forma narracyjna, więc nie wpływały na kształt literatury masowej, ani na strategię odbiorców obcowania z tekstem. Sztandowymi przykładami niesekwencyjności kompozycji dzieł beletrystycznych są *Gra w klasy* (hiszp. *Rayuela*) Julio Cortáзара, czy konceptualne *Sto tysięcy miliardów wierszy* (fr. *Cent mille milliards de poèmes*) Raymonda Queneau. Tak zbudowaną prozę dzieli już tylko krok od gier

fabularnych wykorzystujących książkę jako nośnik fabuły nieliniowej, czyli gier paragrafowych. Pierwszą taką grą była wydana w 1967 roku paragrafówka *Lucky Les* autorstwa Edmunda Hildicka a jedną z pierwszych, jeśli nie pierwszą polską grą paragrafową był *Dreszcz* Jacka Ciesielskiego wydany w 1987 roku.

Nie da się ukryć, że nieliniowe podejście do narracji w utworze jest trudne koncepcyjnie dla autora: musi wymyślić ramy fabularne i taki zestaw wypełniających je całości, żeby każde dozwolone ich połączenie miało sens i było interesujące dla czytelnika. Nie jest to proste. Autor fabuły zazwyczaj tworzy liniową opowieść a jej rozgałęzienia dodaje *post factum*. W tradycyjnym procesie twórczym oznacza to zazwyczaj, że tworzy alternatywne wersje historii połączone z oryginalną kilkoma punktami węzłowymi. *De facto* oznacza to napisanie kilku/kilkunastu utworów i połączenie ich ze sobą. Niewielu autorów jest tak zdeterminowanych, żeby w cenie jednej książki tworzyć kilkanaście, więc klasyczne prozatorskie dzieła o nieliniowej strukturze narracyjnej powstają rzadko i głównie na zasadzie eksperymentu. Nawet niedawne zachwyty nad możliwościami tzw. powieści hipertekstowej nie zaowocowały powszechnością tego typu twórczości.

Ciekawą formą nieliniowości fabuły tworzonej przez wielu autorów jest forma metafabuły wynikającej z zebrania wielu wariantów opowieści stworzonych na określonym terytorium w rozległych ramach czasowych przez wielu niezależnych autorów, którzy jednak mieli wspólne doświadczenia kulturowe, np. znajomość kanonu. Rozpoznawanie ścieżek fabularnych i tworzenie z nich metafabuły jest w takim przypadku domeną nie autorów, którzy nie współpracowali ze sobą i mogli nie mieć pojęcia o swoim istnieniu, ale badaczy i najczęściej dotyczy badań folklorystycznych lub kulturoznawczych paraleli między opowieściami mitycznymi.

Problem z zachowaniem spójności linii fabularnych z regułami świata dotyczy także gier wideo. Problem jest tym większy, że gracz oczekuje swojego realnego wpływu na przebieg fabuły, w odróżnieniu od odbiorcy prozy tradycyjnej, gdzie czytelnictwo zaangażowane nie wyszło poza fazę eksperymentu formalnego. Przed przejściem do dalszego wywodu należy jednak zauważyć, że „wpływ” nie zawsze oznacza wybór rozgałęzień linii fabularnych, w dużej części gier oznacza dążenie do celu po jasno wyznaczonej jednej linii fabularnej z niechcianą opcją zakończenia jej porażką przed osiągnięciem celu.

W każdym z wymienionych przypadków – zarówno tworzenia ścieżek fabularnych w prozie, odkrywania metafabuł na podstawie korpusu istniejących opowieści, jak i projektowania ram fabularnych gier wideo – istotna jest możliwość prześledzenia poszczególnych ścieżek narracyjnych i sposobu, w jaki łączą się ze sobą w punktach węzłowych. Niezbędne jest więc zdefiniowanie najmniejszych całości fabularnych i zaproponowanie narzędzia, aby te całości rozróżniać, układać i sprawdzać, czy łączą się w ścieżkę fabularną.

1.3 Typologia gier komputerowych w kontekście modelu grafowego

Badania groznowcze nie zaowocowały jedną uniwersalną typologią gier. Kategoryzacje, podziały na gatunki lub według cech dystynktywnych tworzone są przez autorów opracowań arbitralnie, na potrzeby omawianych zagadnień.

Jedną z najpopularniejszych wśród twórców gier klasyfikacji jest podział na gatunki zaproponowany przez Ernesta Adamsa (Adams 2011) przy okazji praktycznego opisu problematyki projektowania gier. Jest to klasyfikacja istotna z punktu widzenia zastosowania modelu grafowego, ponieważ obszar zastosowań modelu zamyka się w dwóch z siedmiu wymienionych gatunków (tylko one spełniają warunek obecności narracji, a spośród wszystkich gatunków gier komputerowych interesują nas tylko te gry, w których działania użytkownika wynikają z linii fabularnej).

Klasyfikacja gatunkowa gier wg Ernesta Adamsa

Gry akcji / zręcznościowe (ang. *action games*)

Gry przygodowe (ang. *adventure games*)

Gry logiczne i łamigłówki (ang. *puzzle games*)

Gry fabularne (ang. *role-playing games*)

Gry symulacyjne (ang. *simulation games*), w tym:

- symulacja życia,
- symulacja konstruowania i zarządzania,
- symulatory pojazdów

Gry sportowe (ang. *sports games*)

Gry strategiczne (ang. *strategy games*)

Narrację w stylu tradycyjnej prozy możemy rozpatrywać przede wszystkim w odniesieniu do gier przygodowych. Jest to gatunek gier, w których gracz pełni rolę protagonisty w opowieści, której postęp wyznacza eksploracja świata przedstawionego i wykonywanie zadań, rozwiązywanie stawianych przed nim problemów i zagadek a osiągnięcie głównego celu jest ukoronowaniem całej historii, którą gracz tworzy podczas eksplorowania świata. Fabuły tego typu gier bardzo często inspirowane są literaturą lub filmem, ale zawsze muszą być dopasowane do potrzeb aktywnego uczestniczenia gracza w wyborach głównego bohatera i często są rozbudowane w szereg ścieżek łączących się w kluczowych punktach fabuły. Sztandarowe

przykłady przygodowych gier wideo to seria *Discworld*³ oparta na prozie Terry'ego Pratchetta, seria *The Secret of Monkey Island*, gra *Najdłuższa podróż* (ang. *The Longest Journey*, norw. *Den lengste reisen*) inspirowana prozą Neila Gaimana i twórczością Jossa Whedona, seria *Syberia*, seria *King's Quest* oraz gra *Blade Runner*, bardzo ciekawa pod względem różnorodności zakończeń.

Na potrzeby niniejszej pracy na równych prawach z grami przygodowymi będą traktowane także gry fabularne (RPG)⁴, posiadające rozbudowaną warstwę fabularną. Omawiany tutaj system nie obejmuje reprezentacji formalnej rozbudowanych statystyk postaci będących istotnym elementem gier RPG, ale jak najbardziej nadaje się do modelowania warstwy fabularnej tych gier. Przykładem może być seria *Wiedźmin*, inspirowana powieściami Andrzeja Sapkowskiego, seria *Dragon Age*, czy gra detektywistyczna *Gamedec* inspirowana opowiadaniem Marcina Przybyłka. Wykorzystywanie tych samych motywów fabularnych w różnych typach gier doskonale pokazują gry oparte na opowiadaniu H. P. Lovecrafta *Zew Cthulhu* (ang. *Call of Cthulhu*): wydana w 1981 roku przez studio Chaosium tradycyjna gra fabularna (ang. *pen-and-paper RPG*), gra przygodowa *Call of Cthulhu: Shadow of the Comet* wydana w 1993 roku przez studio Infogrames, strzelanka *Call of Cthulhu: Dark Corners of the Earth* wydana w 2006 roku przez studio Bethesda i wreszcie komputerowa gra fabularna *Call of Cthulhu* wydana w 2018 roku przez studio Cyanide.

W opracowaniach, w których nacisk kładziony jest na związki z literaturą i interakcje z otoczeniem, te dwa gatunki bywają w klasyfikacji łączone (Apperley, 2006), podobnie jak w niniejszej rozprawie.

Należy zwrócić uwagę, że podział na gatunki nie jest sztywny. W dużej części gier akcji istnieje szczątkowa lub bardziej rozbudowana ale pretekstowa fabuła, która pozwala na wyjaśnienie świata początkowego, uzasadnienie cech postaci: wróg, przyjaciel (np. *Wolfenstein*, *Far Cry*). Może wyjaśniać efekty poszczególnych akcji, ale nie wpływa na tworzenie ich długofalowych sekwencji. Tak samo w grach strategicznych elementy fabuły umożliwiają osadzenie rozgrywki w świecie zbliżonym do rzeczywistego (np. gra *Commandos: Behind Enemy Lines* w realiach II wojny światowej) albo w świecie o określonych zasadach (np. gra *Star Wars: Galactic Battlegrounds* w uniwersum *Gwiazdnych Wojen*). W grach innych gatunków czasami

³ W treści rozprawy szczegóły wydania podane są tylko tam, gdzie niezbędne jest rozróżnienie gier o identycznych nazwach, pełne dane bibliograficzne dla każdej gry podane są w spisie omawianych gier na s. 143. Tytuły gier podawane są po polsku dla gier w pełni spolszczonych, w pozostałych przypadkach po angielsku lub w języku oryginału.

⁴ Należy zaznaczyć, że polska nomenklatura jest tu nieco myląca, nie tylko gry fabularne posiadają fabułę, a w ramach tego gatunku fabuła w różnym stopniu wpływa na rozgrywkę (przykładem niewielkiego wpływu są np. gry *hack and slash*, w których istotniejsze od fabuły jest rozbudowanie statystyk postaci i zdobywanie doświadczenia).

występuje też fabuła, ale zazwyczaj ma wtedy znaczenie wyłącznie ornamentacyjne. Należy jednak pamiętać, że podział na kategorie jest płynny, im ważniejsza w rozgrywce gry nieprzygodowej jest fabuła, tym częściej gra wchodzi do kategorii pośredniej, np. *action-adventure*, *action-RPG*.

Najbardziej typowe wykorzystanie elementów fabularnych przedstawione jest poglądowo w tabeli 1, ale należy pamiętać, że można znaleźć na platformach dystrybucyjnych przykłady gier łączących swobodnie cechy dowolnych gatunków.

Tab. 1. Klasyfikacja gatunkowa gier ze wskazaniem obecności i istotności fabuły.

Gatunek	Rola fabuły
Gry akcji	Niekiedy fabuła pozwala na wyjaśnienie świata początkowego, uzasadnia efekty działań gracza.
Gry przygodowe	Najczęściej fabuła jest podstawą dla mechaniki. Związki przyczynowo-skutkowe pomiędzy wydarzeniami mogą być odkrywane w fabularnych interakcjach. Jest bardziej liniowa niż w grach RPG, decyzyjność gracza jest ograniczona węzłami fabularnymi.
Gry logiczne	Jeśli w ogóle fabuła występuje, ma znaczenie ornamentacyjne.
Gry fabularne	Często gry RPG mają świat bardziej otwarty niż typowe gry przygodowe, fabuła pozwala na przewidywanie skutków długofalowych działań i planowanie długofalowych strategii oraz korzystanie z wiedzy ogólnej gracza dotyczącej uniwersum.
Gry symulacyjne	Jeśli w ogóle fabuła występuje, ma znaczenie ornamentacyjne.
Gry sportowe	Jeśli w ogóle fabuła występuje, ma znaczenie ornamentacyjne.
Gry strategiczne	Fabuła prezentuje świat, w którym osadzona jest rozgrywka.

Niezależnie od klasyfikacji gatunkowej można wyróżnić podział na platformy, kategoryzację wiekową, tryb rozgrywki (jedno- lub wieloosobowe, online lub offline), otwartość świata, widok gry. Podziały takie idą w poprzek gatunków, niekiedy zmieniając specyfikę gier w ramach danego gatunku. Na przykład klasyczne gry RPG w wersji papierowej (ang. *pen-and-paper RPG*) są zespołowo tworzoną historią nadzorowaną przez mistrza gry i element wielogłosowości tworzonej opowieści jest w nich bardzo istotny, a w wersji komputerowej początkowo straciły aspekt społeczny, a świat został sparametryzowany pod rolę

komputera jako mistrza gry. W literaturze niekiedy stosuje się rozróżnienia akronimów: CRPG dla komputerowych gier fabularnych (w odróżnieniu od tradycyjnych) a MMORPG dla gier wieloosobowych (ang. *massively multiplayer online role-playing game*). Rozróżnienie to traci jednak coraz bardziej na popularności.

Podział na gatunki nie pomoże nam jednak zaproponować kategoryzacji w ramach zbioru gier reprezentowanych za pomocą modelu grafowego, niewiele mówi bowiem o cechach gier wpływających na kształt tego modelu. Aby pokazać dopasowanie struktury grafowej do formy gry, lepiej jest sięgnąć po klasyfikację gier pod kątem cech dystynktywnych niż po tradycyjne gatunki. Posłużmy się tutaj kategoryzacją Espena Aarsetha (Aarseth, 2003), który wyróżnił cechy dystynktywne w czterech kategoriach.

Klasyfikacja Aarsetha została przedstawiona poniżej wraz przykładami gier z dwóch gatunków będących przedmiotem zainteresowania autorki niniejszej rozprawy w kontekście zastosowania modelu grafowego. Zaznaczono pogrubieniem te cechy gier, których odwzorowanie doskonale wpasowuje się w możliwości modelu grafowego.

Wielowymiarowa kategoryzacja gier wg Espena Aarsetha

Przestrzeń

1. Perspektywa: globalna (ang. *omnipresent*) / **lokalna** (ang. *vagrant*)

Zarówno typowe gry przygodowe (np. *Discworld*) jak i typowe gry fabularne (np. *Wiedźmin*) korzystają z perspektywy lokalnej.

2. Topografia: geometryczna / **topologiczna**

Charakterystyczne dla gier przygodowych jest topologiczne podejście do przestrzeni (np. gra *Teenagent*, jedna z pierwszych polskich gier przygodowych), inaczej wygląda sytuacja w przypadku gier RPG, gdzie to podejście jest najbardziej charakterystyczne dla podgatunku *dungeon crawl*, np. *Legend of Grimrock 1, 2* lub *Dungeon Master 2*, a częściej spotykane jest ujęcie geometryczne, np. *Assassin's Creed*.

3. Zmiany otoczenia: **statyczne** / dynamiczne

Możliwość wpływania na lokację przez gracza jest bardzo ograniczona w grach przygodowych i sprowadza się do specjalnie zaprojektowanych interakcji (np. *The Secret of Monkey Island*) podczas gdy w grach RPG częściej mamy do czynienia z elementami świata otwartego, co bywa wykorzystywane przez graczy jako nadużycie fabularne (ang. *exploit*), z czego znane są gry *The Elder Scrolls IV: Oblivion*, czy *The Elder Scrolls V: Skyrim*.

Czas

4. Następstwo wydarzeń (ang. *pace*): gry czasu rzeczywistego / **gry turowe**

Gry przygodowe w większości są beczasową serią stanów dyskretnych, niezwykle rzadko zdarzają się gry czasu rzeczywistego (np. *Blade Runner*). Sytuacje interakcji z bohaterami niezależnymi zbudowane bywają w sposób turowy, np. dialogi. Gry RPG natomiast bardzo często odbywają się w czasie rzeczywistym (np. *Skyrim*), turowo bywają poprowadzone tam walki, a niekiedy można w opcjach gry wybrać formę walki albo w czasie rzeczywistym albo turowym (np. *Fallout 4*). Starsze gry częściej bywają zaprojektowane w pełni turowo (np. *Fallout 1, 2*).

5. Upływ czasu (ang. *time representation*): mimetyczny / **arbitralny**

Następstwem konstrukcji beczasowej gier przygodowych jest to, że trwanie poszczególnych zdarzeń, jako pozbawione wpływu na rozgrywkę, jest w pełni arbitralne. W nowszych grach RPG (np. *Oblivion*, *Skyrim*, *Fallout 3, 4*) dynamika zdarzeń i czas ich trwania zwykle wynika z obliczeń silnika fizyki.

6. Celowość: **osiągalny cel** (ang. *clear winning state*) / nieskończona gra (ang. *endless game*)

Immanentną cechą gier fabularnych jest dążenie gracza do dobrze określonego celu, choć nie jest powiedziane, że w trakcie rozgrywki nie nastąpi jego konwersja. Osiągnięcie celu jest w typowej grze przygodowej zakończeniem gry. Natomiast gry RPG pozwalają na eksplorowanie świata niezależnie od ewentualnie zdefiniowanego celu a także kontynuowanie rozgrywki po osiągnięciu tegoż. Szczególnie wyraźnie widać to w grach MMORPG, gdzie zakres interakcji pomiędzy postaciami jest praktycznie nieograniczony (np. *World of Warcraft*).

Kooperacja (ang. *player structure*)

7. Relacje między postaciami: **gra jednoosobowa** / gra wieloosobowa

Typowe gry przygodowe projektowane są z myślą o pojedynczym gracz, niekiedy mamy do czynienia z wyjątkami, np. gra *Brothers: A Tale of Two Sons*. Wśród gier fabularnych częściej zdarzają się wieloosobowe, np. *Assassin's Creed*, *World of Warcraft*, *Diablo*, *Borderlands*.

Kontrola

8. Rozwój (ang. *mutability*): brak / **wzmocnienia** (ang. *powerups*) / rozwijanie postaci (ang. *experience-leveling*)

Możliwości, jakie mamy w grach fabularnych, zazwyczaj związane są z konkretnymi zadaniami i okolicznościami pozwalającymi na ich wykonanie, możemy więc mówić

o czasowych wzmocnieniach postaci. W grach fabularnych istotnym elementem rozgrywki jest budowanie statystyki postaci, której rozwój jest progresywny a możliwość wykorzystania zdobywanych umiejętności nie ogranicza się do jednego zadania. Często wiąże się to z wprowadzeniem rozbudowanej ekonomii świata. Przykładem pierwszego podejścia jest np. gra *The Legend of Kyrandia: Hand of Fate*, gdzie główną bohaterką jest czarodziejka, która w określonych momentach może przygotować eliksir umożliwiający wykonanie konkretnego działania, a drugiego *Wiedźmin*, którego tytułowy bohater rozwija umiejętności przyrządzania różnych eliksirów, które może wykorzystać w dowolnym momencie i w różny sposób (rozbudowana ekonomia pozwala np. handlować stworzonymi przez siebie artefaktami).

9. Możliwość wstrzymania gry: **w dowolnym momencie** / w wyznaczonych punktach / brak
- Filozofia wstrzymania biegu rozgrywki jest różna w zależności od podejścia do upływu czasu i od utraty możliwości dalszej rozgrywki. Typowa gra przygodowa jest odwzorowana przez bezczasowy stan świata a w dodatku projektanci unikają postawienia gracza w sytuacji bez wyjścia. Możliwość zapisania gry i późniejszego jej odtworzenia występuje praktycznie w każdym momencie, ale nie ma strategicznego znaczenia dla gracza. Przykładem ciekawej choć nietypowej zachęty do długich sesji gry bez zapisu jest polska gra *Lords of the Fallen*, gdzie ciągłość gry bez zapisu daje większe nagrody w przypadku sukcesu danego działania. W grach RPG możemy odtworzyć stan gry z wybranych kluczowych momentów, na specjalną uwagę zasługują gry nie dające nawet takiej możliwości, np. gra *Diablo* w trybie *Hell*.

10. Przewidywalność: **gra deterministyczna** / gra niedeterministyczna

Charakterystyczną cechą gier przygodowych jest determinizm działań w świecie, podczas gdy w grach fabularnych częściej położony jest nacisk na pewną losowość wydarzeń (np. *Fallout*, *Skyrim*).

11. Reguły: **topologiczne** / czasowe / **obiektywne**

Typowe dla obu gatunków omawianych gier są reguły bazujące na topologii przestrzeni lub na relacji pomiędzy obiektami. Reguły czasowe niekiedy mają zastosowanie w grach RPG w sekwencjach walki.

Cechy dystynktywne opisane w powyższej kategoryzacji znakomicie ułatwiają porównawczą charakterystykę istniejących gier jak i są podstawą do określania założeń projektowych gier nowo tworzonych.

1.4 Struktury narracyjne w teorii literatury

Typologia struktur fabularnych i budowanie metateorii narracyjnych jest dziedziną badań o długiej tradycji i jest nie tylko przedmiotem badań literaturoznawców, ale istotnie wpływa na proces tworzenia dzieła literackiego przez autorów. Praktycznie już pierwsze zapisane teksty prozatorskie wskazywały na dużą świadomość formalną ich twórców (możemy wskazać tu przykład *Don Kichota*, dzieła wczesnego, ale już niezwykle autoreferencyjnego narracyjnie). Rozwój teorii form fabularnych szedł w parze z rozwojem gatunków prozatorskich i dramatycznych. Współcześnie mamy wiele teorii narracji, za większością z nich stoi próba typologii a za niektórymi próba formalizacji struktur narracyjnych dzieła literackiego. Formalizacja jest możliwa szczególnie w gatunkach, gdzie występuje silna konwencjonalizacja fabularna. Niektóre dziedziny sztuki fabularnej są bardziej podatne na siłę konwencji niż inne. Silna konwencjonalizacja opowieści widoczna jest w strukturach narracyjnych opowieści mitycznych, bajkach, kryminałach, ale także strukturach interaktywnych, czy w szczególności grach wideo, przedmiocie zainteresowania autorki niniejszej rozprawy. W każdym z tych typów narracyjnych można pokusić się o formalizację opowieści w oparciu o funkcje pełnione przez bohaterów opowieści. Formalizacja struktur gry wideo może opierać się o koncepcje literaturoznawcze, które uwzględniają problem wariantowości fabuły. Spośród wielu takich modeli teoretycznych jednym z najciekawszych jest model rosyjskiego strukturalisty Władimira Proppa, który zajmował się badaniami bajki magicznej. Okazuje się, że kanon bajek ze swoją wariantowością może być podstawą do analizy różnorodności fabularnej wariantów gry wideo, dlatego ta właśnie teoria stała się inspiracją nie tylko niniejszej pracy, ale wielu systemów generowania fabuł. Przyjrzyjmy się jej bliżej.

Władimir Propp był rosyjskim literaturoznawcą i formalistą. Zajmował się w swoich badaniach strukturami narracyjnymi, największą sławę przyniosła mu praca *Морфология сказки* (pol. *Morfologia bajki*), szczególnie po odkryciu jego publikacji przez Claude'a Lévi-Straussa i krąg francuskich strukturalistów (Lévi-Strauss, 1968).

Celem Władimira Proppa była klasyfikacja i opis morfologiczny bajki magicznej. Zwróćmy uwagę, że spośród całego spektrum utworów dających się określić mianem bajki badacz zdecydował się stworzyć teorię dotyczącą pewnego wycinka owego spektrum. „Bajka magiczna posiada specyficzną konstrukcję, konstrukcję wyczuwaną od razu, choć często nieświadomie.” – tłumaczył.

Metoda Proppa polega na interfabularnym porównaniu bajek. W tym celu wydziela konstytuujące je części, posługując się przedstawionymi w pracy metodami. Następnie porównuje

bajki ze względu na tak wydzielone części. W rezultacie otrzymuje morfologię, tj. opis bajki według jej części składowych, stosunków wzajemnych zachodzących między nimi oraz ich odniesień do całości (Propp, 1968, s. 206).

Jakimi metodami można osiągnąć dokładność takiego opisu? Porównajmy następujące przypadki:

- 1) Car ofiarowuje junakowi orła. Orzeł przenosi junaka do innego królestwa.
- 2) Dziad daje Suczence konia. Koń przenosi Suczenkę do innego królestwa.
- 3) Królowna ofiarowuje Iwanowi pierścień. Junacy mieszkający w pierścieniu przenoszą Iwana do innego królestwa.
- 4) Czarodziej daje Iwanowi łódkę. Łódka przenosi Iwana do innego królestwa.

W przytoczonych przykładach dane są wielkości stałe i zmienne. Zmieniają się nazwy (a wraz z nimi atrybuty) osób działających, nie zmieniają się natomiast ich działania, czyli funkcje. Stąd wniosek, że bajka często przypisuje identyczne działania różnym postaciom. Fakt ten pozwala na badanie jej w oparciu o funkcje działających postaci. (...) Tak więc funkcje działających postaci są podstawowymi elementami bajki i te właśnie elementy winniśmy tu wyodrębnić.

W toku wywodu Propp decyduje, że określając funkcję nie liczymy się z jej realizatorem i identyfikujemy funkcję przez rzeczownik określający czynność (np. zakaz, wypytywanie, ucieczka). Zaznacza też konieczność określenia jej kontekstu, czyli znaczenia w ciągu kolejnych akcji.

Funkcje stanowią podstawowe części składowe bajki. Ich liczba jest ograniczona. Propp zidentyfikował trzydzieści jeden podstawowych funkcji, ale dla wielu z nich wyodrębnił warianty pochodne, np. ogólne odejście, bardziej szczegółową śmierć członka rodziny i jeszcze bardziej szczegółową śmierć rodziców.

Materiał badawczy będący podstawą pracy Władimira Proppa został dobrany zgodnie z klasyfikacją Aarne-Thompsona. Badacz ograniczył się do zakresu 300–499 z 2700 typów podzielonych na cztery główne kategorie.

Struktura narracyjna bajki definiowana jest następująco. Analizę Propp rozpoczyna od zdefiniowania sytuacji wyjściowej, w przypadku gier wideo analogią będzie początkowy stan świata (Propp, 1968, s. 209):

Bajka rozpoczyna się zazwyczaj od pewnej sytuacji wyjściowej (wyliczenie członków rodziny, wprowadzenie przyszłego bohatera poprzez wymienienie jego imienia albo wzmianki o sytuacji, w jakiej się on znajduje). Chociaż sytuacja ta nie jest funkcją, stanowi jednak ważny element morfologiczny. (...) Określimy ów element jako sytuację początkową.

Następnie definiuje wspomniane wcześniej trzydzieści jeden funkcji nakreślając ich warianty rozumiane jako funkcje pochodne, uszczegółowione w stosunku do pierwotnych. Przypisuje każdej funkcji symbol a hierarchię zależności oznacza numerami, na przykład:

Odejście: e . Odejście rodziców (może być niedefinitywne: do pracy, do lasu, na targ, na wojnę itp.) to: e^1 . Spotęgowaną formą odejścia jest śmierć rodziców: e^2 .

Następnie określa postaci w fabule. Głównego bohatera definiuje poprzez cel, który ów chce osiągnąć, co też daje się przełożyć na analogie celu gry (Propp, 1968, s. 213):

Bohater bajki magicznej jest to taka postać, która albo bezpośrednio w zawiązaniu akcji poniosła stratę wskutek działań przeciwnika (resp. odczuwająca pewien brak), albo też taka, która zgodziła się zlikwidować nieszczęście (lub brak) dotyczące innej osoby.

Pozostali bohaterowie definiowani są poprzez role, które pełnią pomagając lub przeszkadzając bohaterowi ten cel osiągnąć i możliwości, jakie w tym zakresie posiadają. W sumie mamy siedem kategorii zwanych przez Proppa kręgami akcji:

1. bohater,
2. przeciwnik,
3. donator (osoba poddająca bohatera próbie),
4. pomocnik,
5. królewna lub jej ojciec,
6. osoba wysyłająca,
7. fałszywy bohater.

Zestawienie użytych funkcji i bohaterów je inicjujących pozwala zapisać każdą z analizowanych bajek jako formułę składającą się z symboli kodujących funkcje i ich realizatorów (por. Propp 1968, s. 226; Propp 1968, s. 227).

W przypadku tworzenia modeli wspomagania projektowania systemów tworzenia fabuł (a przykładem takiego systemu jest gra wideo traktowana jako generator rozgrywek, czyli sekwencji fabularnych) model Proppa trzeba potraktować jako inspirację a nie fundament działania, co nie zmienia faktu, że jest to niezwykle inspirujący model. Współczesne literaturoznawstwo odeszło od poszukiwania wielkich narracji jak i od strukturalizmu jako formy opisu rzeczywistości, natomiast w systemach generacyjnych, mniej nastawionych na typologię a bardziej na zastąpienie lub wspomnienie kreatywności autorów i zapewnianie spójności wewnętrznej fabuły – strukturalne podejście przeżywa rozkwit.

Systemy generacyjne fabuł i quasifabuł dotyczą przede wszystkim krótkich form narracyjnych w określonych ramach formalnych i tematycznych (np. generowanie scenariuszy odcinków opery mydlanej lub generowanie historyjek kryminalnych), ale także po prostu sekwencji akcji postaci, generowania dialogów. Efekty działania takich systemów zależne są oczywiście nie tylko od modelowania struktury narracyjnej, ale zastosowania mechanizmów i algorytmów tworzenia fabuł: wykorzystują np. sztuczną inteligencję lub systemy agentowe. Jako przykłady różnorodnych zastosowań można przytoczyć prace S. Imabuchiego i T. Ogaty *Story Generation System Based on Propp Theory as a Mechanism in Narrative Generation System* z 2021 roku, S. Malca *Proppian Structural Analysis and XML Modeling* z 2001 roku czy T. Wamy i R. Nakatsu, *Analysis and Generation of Japanese Folktales Based on Vladimir Propp's Methodology* z 2008 roku. Na szczególną uwagę zasługuje model „Afanasyev” wspomagający zespołowe zautomatyzowane tworzenie fabuł i prace z tego zakresu naukowców skupionych wokół profesora Uniwersytetu Madryckiego (Universidad Complutense de Madrid), Pablo Gervása. Przykładowe prace jego zespołu z ostatnich kilku lat obejmują: *Assessing MultiPlot Stories: from Formative Analysis to Computational Metrics*, *Exploring Baselines for Combining Full Plots into Multiple-plot Stories*, *Quantitative Characteristics of Human-Written Short Stories as a Metric for Automated Storytelling*, *Empirical Insights Into Short Story Draft Construction* z 2020 roku, *Afanasyev: A Collaborative Architectural Model for Automatic Story Generation* z 2018 roku.

Zbieżności między elementami teorii Władimira Proppa a zaproponowanym w pracy modelem struktury fabularnej gry wideo

Podobnie jak u Władimira Proppa w niniejszej rozprawie skupiono się na strukturze pewnego wycinka twórczości, jaką są gry wideo. Stworzenie systemu odwzorowującego strukturę fabularną dowolnej gry wideo wykracza poza ramy tej pracy – arbitralnie ograniczono się do gatunku gier przygodowych. Mają one skądinąd w sobie istotne analogie do logiki przyczynowo-skutkowej następstw wydarzeń w bajkach magicznych.

Zwróćmy uwagę na analogię między wariantami baśni a różnymi rozgrywkami pojedynczej gry. W przypadku badania folkloru wariantywność bajek określa się *a posteriori* na podstawie zebranych świadectw a struktura narracyjna wyłania się z ich porównania w oparciu o zaproponowaną typologię. W przypadku tworzenia gry wideo a w szczególności projektowania jej mechaniki, ostatecznym twórcą fabuły jest gracz (odpowiada on osobie bajora ludowego przedstawiającego swoją wersję historii) ale ramy narracyjne trzeba wcześniej zaprojektować

i zakodować – i tym zajmują się twórcy gry, czyli struktura narracyjna musi być zdefiniowana *a priori*, choć nie zawsze projektanci gry mają świadomość znaczenia swoich wyborów implementacyjnych dla różnorodności hipotetycznych rozgrywek.

Sytuację wyjściową w prezentowanym w rozprawie systemie reprezentuje graf stanu świata, choć w odróżnieniu od Proppa autorka rozprawy nie buduje typologii stanów inicjacyjnych. Za to na jego koncepcji funkcji, w szczególności ich hierarchizacji w zależności od szczegółowych wariantów wzorowano się proponując hierarchię transformacji modyfikujących stan świata i koncepcję ich uszczegóławiania. Oderwanie funkcji od realizatora osiągnięto dzięki wprowadzeniu pojęcia produkcji generycznych. Kontekst, który dla rosyjskiego badacza jest bardzo ważny dla interpretacji poszczególnych działań, jest odwzorowywany w grafie rozgrywki i ostatecznie każdą rozgrywkę można zakodować podając sekwencję zmian stanu świata gry.

Odwzorowania tych koncepcji w modelu formalnym są opisane w rozdziale trzecim a szczegółowe rozwiązania implementacyjne w rozdziałach kolejnych.

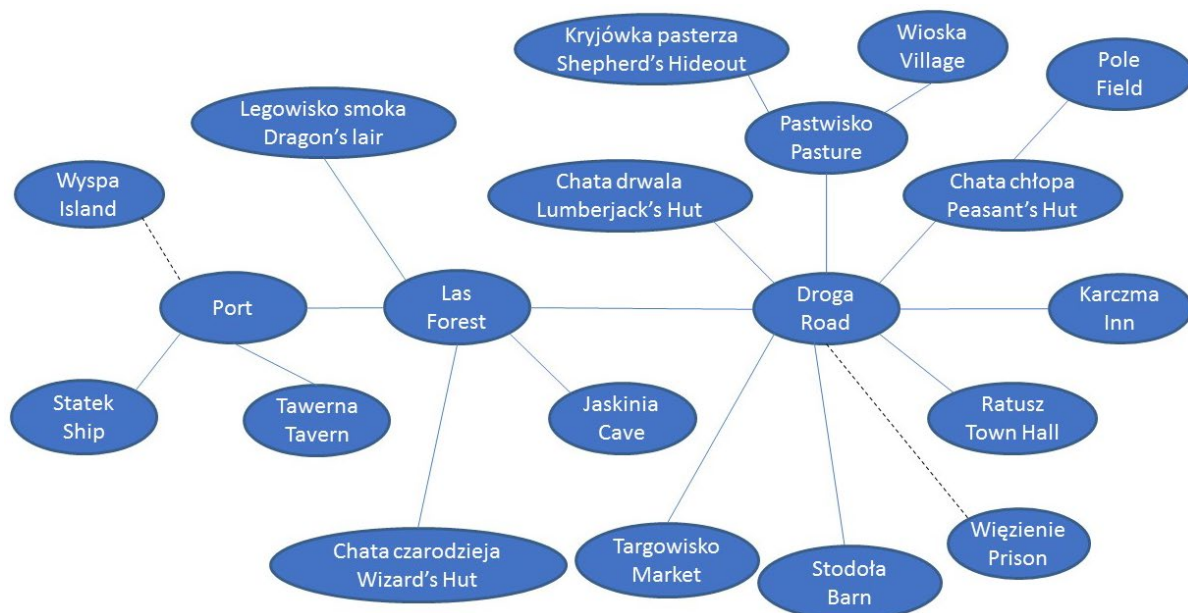
2 Studium przypadku

Wszystkie przykłady pokazane w niniejszej pracy będą opierać się na systemie fabularnym gry przygotowanym przez autorkę na potrzeby zajęć dydaktycznych z przedmiotu „Projektowanie wspomagane komputerem” i rozwijanym przez studentów w latach 2019–2022 w projekcie fabularnym „Royal Envoy”. System został zaimplementowany przy użyciu języka Python 3.9. Oprogramowanie wspomagające dynamiczne tworzenie animacji w oparciu o powyższy model było tworzone przez studentów przy użyciu silnika Godot 3. Grafiki będące podstawą animacji zostały także stworzone przez studentów.

2.1 Fabuła gry przykładowej

Fabuła rozgrywa się w świecie zawierającym dwadzieścia lokacji (por. rys. 1). Głównym bohaterem jest kurier, którego zadaniem jest przewieźć otrzymane tajne dokumenty do zamorskiego królestwa. Statek tam płynący to lokacja końcowa, której osiągnięcie wraz ze wzmiankowanymi dokumentami oznacza wygraną. Przegrana następuje w momencie śmierci bohatera. W grze występują także postaci niezależne (NPC, ang. *non-player character*). W trakcie rozgrywki nie można tworzyć, usuwać ani zmieniać relacji między lokacjami, są to działania dostępne wyłącznie dla projektantów podczas tworzenia gry. Zasada ta nie dotyczy postaci i przedmiotów – istnieją działania, które skutkują powstaniem nowych postaci i przedmiotów, będących reprezentantami określonego typu identyfikowanego etykietą. Zbiór dostępnych etykiet jest zamknięty.

Nie można tworzyć obiektów o etykietach spoza tego zbioru. Każdy typ obiektu skojarzony jest z symboliczną grafiką (przykłady przedstawione są na rysunkach 2 i 3).



Rys. 1. Mapa lokacji w systemie gry. Krawędzie przerywane reprezentują lokacje dostępne pod pewnymi warunkami.

Możliwe działania postaci zapisywane są w formacie JSON zgodnym z modelem grafowym opisanym w kolejnych rozdziałach. Można je tworzyć w dowolnym zewnętrznym edytorze tekstowym. Działania mają charakter generyczny (dotyczący każdej postaci w dowolnych okolicznościach) lub szczegółowy (specyficzne działania służące osiągnięciu określonego celu). Akcje są zgrupowane w misje (ang. *quest*), co jest przydatne podczas procesu projektowego, mogą być jednak wykorzystane w dowolnym momencie rozgrywki, jeżeli tylko stan świata gry na to pozwala. Poszczególne akcje układają się w hierarchę zależności rozumianą jako wyrowadzenie działania bardziej szczegółowego z działania o charakterze ogólnym.



Rys. 2. Przykładowe postaci w grze.



Rys. 3. Przykładowa lokacja.

W systemie rozpatruje się akcje inicjowane przez gracza (czyli wykonywane przez postać, w którą wciela się gracz), akcje wykonywane przez bohaterów niezależnych (poprzez system sterowania postaciami niezależnymi) oraz akcje automatyczne (dziejące się w świecie zawsze po zaistnieniu pewnych warunków, nie mające inicjatora).

System pozwala na przeprowadzenie symulacji procesu decyzyjnego gracza, wizualizację wybranych ścieżek fabularnych a także na analizę fabuły poprzez generowanie potencjalnych ścieżek rozgrywki w poszukiwaniu problemów: niekonkluzywnych zakończeń, pętli, zbyt szybkiego osiągnięcia końca gry etc. Możliwości analizy struktury fabularnej opisane są w rozdziale 6.1. Pomocniczymi funkcjami systemu jest sprawdzanie poprawności struktur narracyjnych i ich wizualizacja, budowanie światów i hierarchii akcji. Możliwości referencyjnej implementacji opisane są w rozdziale 5.4.

3 Model grafowy

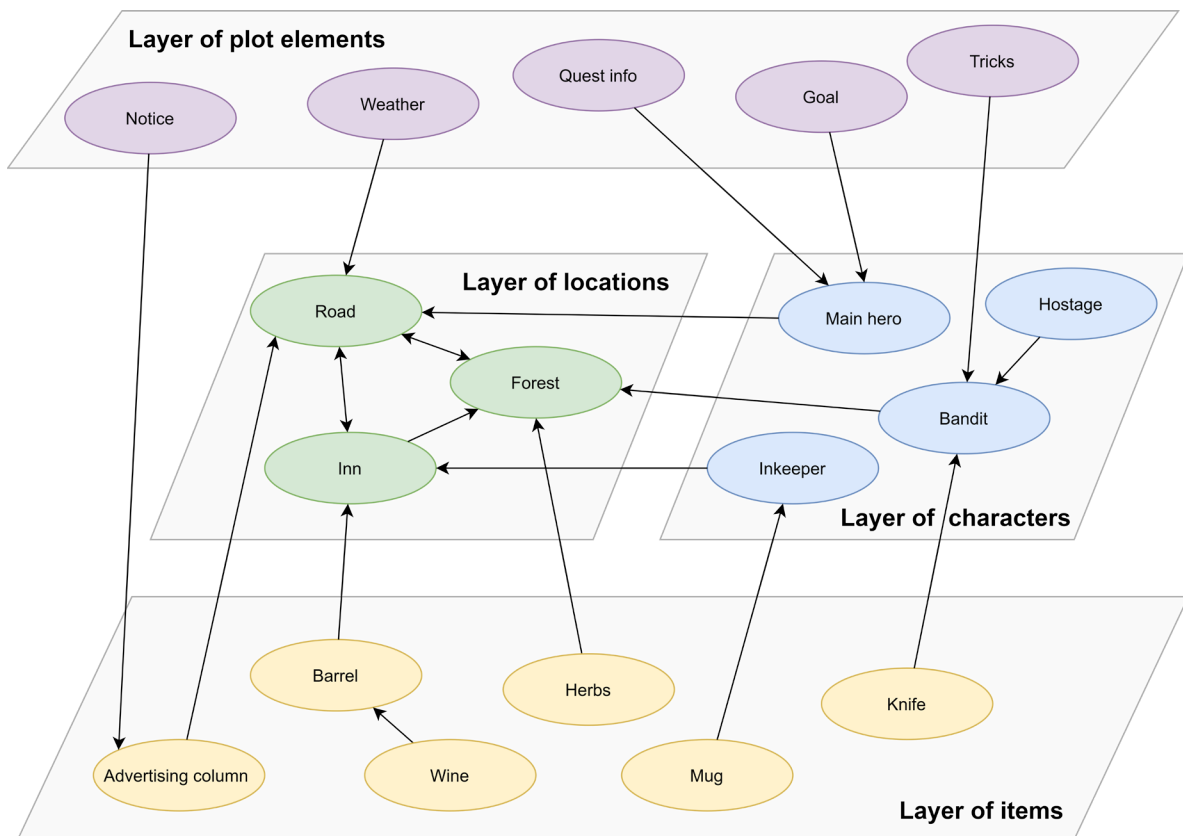
Model narracyjny gry fabularnej odwzorowuje istotne obiekty świata, relacje i zależności pomiędzy nimi. Podstawą modelu fabuły jest graf zwany dalej grafem stanu świata gry (GWSG, ang. *Game World State Graph*), który opisuje aktualny stan świata i jest modyfikowany po każdym ruchu gracza i każdym działaniu postaci niezależnej.

Graf ten składa się z wierzchołków odpowiadających obiektom świata gry i krawędzi reprezentujących relacje między tymi obiektami. Pod względem charakterystyki możemy wyróżnić cztery typy wierzchołków: odpowiadające lokacjom, bohaterom, przedmiotom oraz elementom wiedzy narracyjnej. Każdy typ wierzchołków wraz z relacjami między nimi tworzy warstwę. Rysunek 4 przedstawia przykładowy GWSG z zaznaczonymi czterema warstwami⁵.

Krawędzie tego grafu są skierowane. Ich znaczenie dla fabuły związane jest z tym, czy są to krawędzie wewnętrzne reprezentujące relacje pomiędzy wierzchołkami z danej warstwy, czy krawędzie reprezentujące relacje pomiędzy wierzchołkami z różnych warstw.

Atrybuty traktujemy jako funkcje zdefiniowane na obiektach, których przeciwdziedziny określają zakres wartości zależny od własności obiektu, którą atrybut opisuje. Różne atrybuty określone są dla różnych typów wierzchołków warstw.

⁵ Ze względu na to, że duża część ilustracji bazuje na materiałach z gry „Royal Envoy”, w której wszystkie obiekty świata opisane są w języku angielskim, teksty na ilustracjach wykorzystują ten język. Ich polskie odpowiedniki znajdują się w głównym tekście pracy.



Rys. 4. Graf stanu świata gry (GWSG) z wyróżnionymi czterema warstwami.

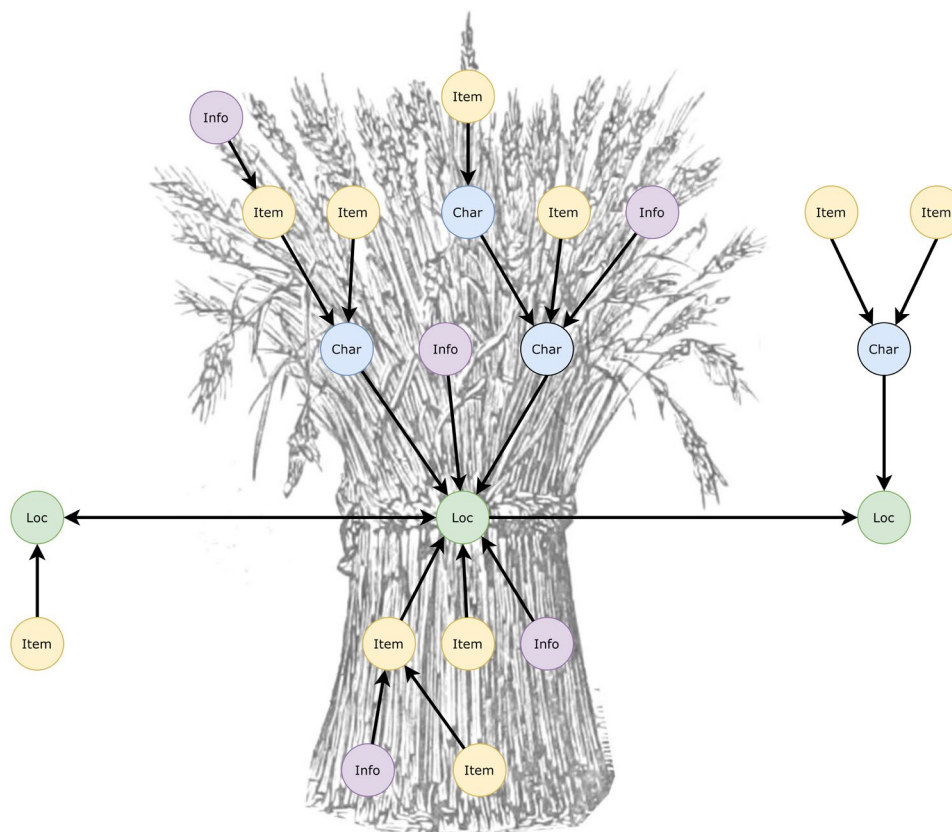
Snopkowa struktura danych

W celu optymalizacji wykonywania działań typowych dla mechanik gier wideo zaproponowano nową grafową procedurę generacyjną polegającą na transformowaniu grafów według formalnych reguł zwanych produkcjami. Implementacja zastosowania produkcji dla gier wideo wymaga użycia odpowiedniej struktury danych opartej o autorską koncepcję podgrafów snopkowego i półsnopkowego.

Podgraf snopkowy (ang. *sheaf subgraph*) to podgraf zawierający jedną lokację i węzły z innych warstw połączone z nią bezpośrednio lub pośrednio. Podgraf snopkowy dobrze obrazuje wycinek świata dostępny bieżącej percepcji gracza.

Podgraf półsnopkowy to podgraf zawierający węzeł niebędący lokacją i węzły z innych warstw połączone z nim bezpośrednio lub pośrednio. Zdefiniowanie grafów półsnopkowych pozwala na bardzo prostą implementację najczęściej wykonywanych w grze operacji na węzłach.

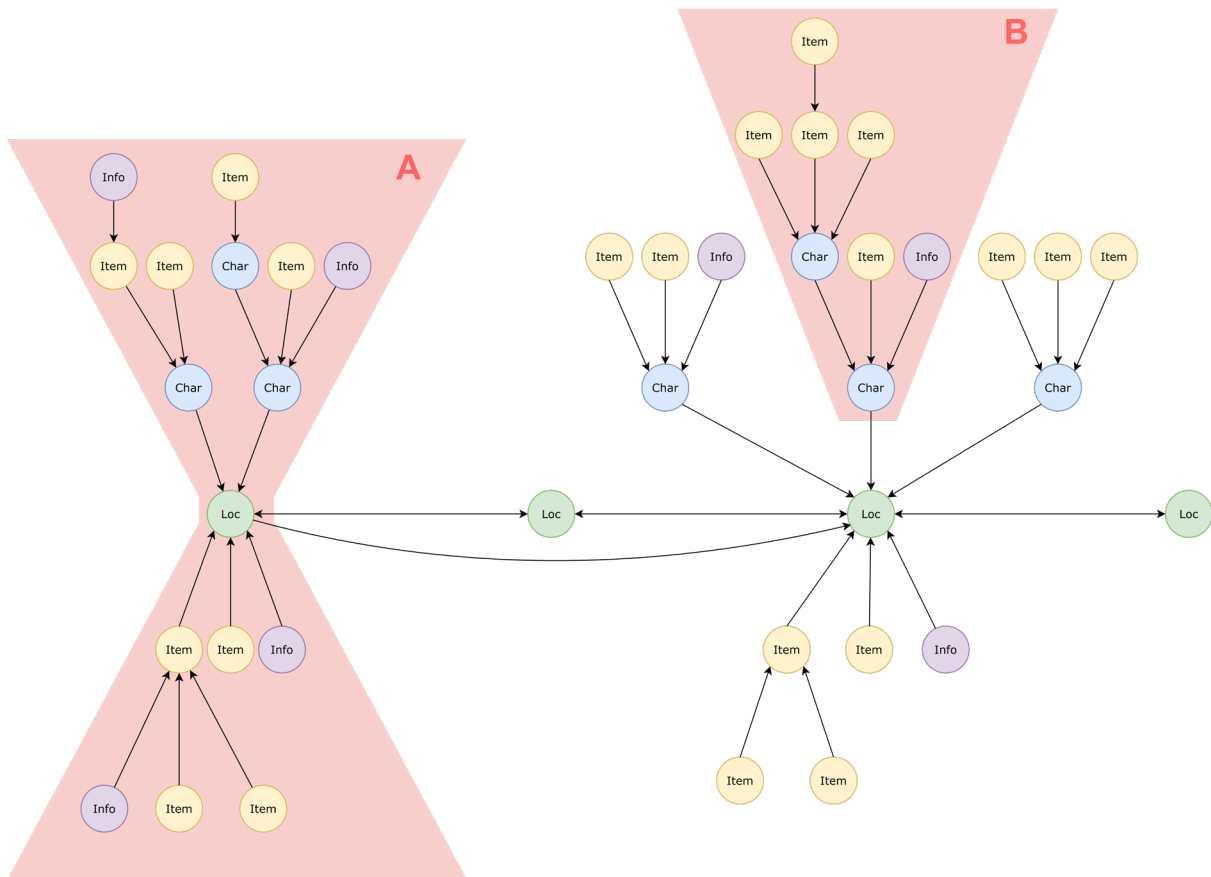
Ideę grafu snopkowego symbolicznie zaprezentowano na rysunku 5 a przykład wyróżnionych w grafie struktur podgrafu snopkowego i półsnopkowego przedstawiono na rysunku 6.



Rys. 5. Idea grafu snopkowego.

Cały graf gry można traktować jako las podgrafów snopkowych osadzonych w podgrafie warstwy lokacji poprzez korzenie-lokacje. Wszystkie ścieżki wychodzące z danej lokacji rozgałęziają się, tworząc graf snopkowy. Prowadzi to do definicji grafu wielosnopkowego (por. rozdział 3.1, *Definicja 8*). Typowe modyfikacje świata wynikające z mechaniki gry najczęściej operują na jednym lub kilku snopkach.

Zastosowanie struktury snopków konsekwentnie dla grafu umożliwia zastosowanie specyficznego zagnieżdżonego zapisu grafu w formacie JSON, w którym przechowywane są informacje o grafie (por. rozdział 4.5). Struktura snopkowa i umożliwiony przez nią zapis JSON, odbiegający od typowych reprezentacji grafowych struktur danych, został zaprojektowany pod kątem optymalizacji działań na grafie wynikających z mechanik gier fabularnych.



Rys. 6. Podgraf snopkowy (A) oraz półsnopkowy (B) w strukturze GWSG.

Modyfikacja grafu stanu świata gry

Niezależnie od tego, czy gra reprezentowana w modelu jest grą turową, czy grą czasu rzeczywistego, podstawą rozgrywki są wydarzenia dziejące się w świecie gry. Wydarzenia mogą być inicjowane przez gracza, przez postaci niezależne lub wynikać z logiki funkcjonowania świata (zmiana pór dnia, utrata sił życiowych przez bohaterów etc.). Typy możliwych wydarzeń narzuca mechanika gry. Aby można było mówić o modelu grafowym gry, każde wydarzenie należy opisać za pomocą produkcji grafowej procedury generacyjnej.

Warunki stosowalności. Nie zawsze samo istnienie podgrafu o określonej strukturze i mających wierzchołki o określonych własnościach wystarczy do wykonania akcji. Takie dodatkowe ograniczenia znane są w literaturze jako warunki stosowalności albo predykaty stosowalności (por. Ehrig, Habel 1986). Odwołują się one do wartości atrybutów w wybranym podgrafie i zwracają prawdę albo fałsz. Niespełnienie ich pozwala odrzucić produkcję mimo dopasowania lewej strony produkcji do grafu.

Instrukcje. W opisywanym modelu grafowym przedsięwzięcie akcji sprowadza się do wykonania ciągu instrukcji operujących na wierzchołkach wybranego podgrafu i strukturach snopkowych zbudowanych w oparciu o te wierzchołki.

Poziom ogólności. Produkcje możemy charakteryzować według różnych kryteriów, ale bardzo istotną cechą jest własność generyczności oparta o formalne mechanizmy pozwalające definiować akcje w sposób bardziej ogólny, tzn. nie ograniczając ich stosowania do węzłów o określonych etykietach.

Hierarchizacja akcji. Konsekwencją sformalizowania akcji gracza w postaci produkcji grafowych jest możliwość ich hierarchizacji, czyli wprowadzenia częściowego porządku w zbiorze produkcji. Taka hierarchia produkcji jest rozszerzeniem modelu grafowego przedstawionego formalnie w rozdziale 3.1 o graf hierarchii produkcji. Można go traktować jako kolejny poziom opisu fabuły:

1. statyczny opis stanu świata za pomocą GWSG,
2. zdefiniowanie zmian tego stanu za pomocą zbioru produkcji,
3. częściowe uporządkowanie zbioru produkcji decydujące o priorytecie ich stosowania.

3.1 Formalne definicje grafowe

Niech L będzie alfabetem etykiet wierzchołków, tzn. skończonym zbiorem niepustym.

Definicja 1 – Etykietowany graf skierowany

Etykietowanym grafem skierowanym jest system:

$$C = (V, E, s, t, lab), \quad (1)$$

gdzie:

- V oraz E są skończonymi rozłącznymi zbiorami odpowiednio wierzchołków i krawędzi,
- $s : E \rightarrow V$ jest funkcją źródłową krawędzi,
- $t : E \rightarrow V$ jest funkcją docelową krawędzi,
- $lab : V \rightarrow L$ jest funkcją etykietowania wierzchołków.

Wprowadźmy następujące oznaczenia $V_C, E_C, s_C, t_C, lab_C$ na elementy grafu C .

Przykład 1

Zdefiniujmy teraz cztery grafy odpowiadające powyższej definicji. Ich symboliczne wizualizację przedstawimy na rysunkach 7, 8, 9, 10.

Graf lokacji

Niech L_{LOC} będzie alfabetem nazw lokacji.

Grafem lokacji będzie system $LOC = (V_{LOC}, E_{LOC}, s_{LOC}, t_{LOC}, lab_{LOC})$, gdzie:

V_{LOC} – wierzchołki reprezentujące lokacje,

E_{LOC} – krawędzie wskazujące dostępność lokacji za pomocą funkcji s_{LOC}, t_{LOC} ,

lab_{LOC} – funkcja przypisująca nazwy lokacji wierzchołkom.

$L_{LOC} = \{\text{Dragons_lair, Forest, Inn, Pasture, Road, Village, Wizards_hut}\}$

$V_{LOC} = \{v_1, v_4, v_6, v_7, v_{13}, v_{14}, v_{16}\}$,

$E_{LOC} = \{e_1, e_4, e_6, e_7, e_{13}, e_{14}, e_{16}, e_{25}, e_{26}, e_{27}, e_{28}, e_{29}, e_{30}, e_{31}\}$

$lab_{LOC}(v_1) = \text{Wizards_hut}$, $lab_{LOC}(v_4) = \text{Dragons_lair}$, $lab_{LOC}(v_6) = \text{Forest}$, $lab_{LOC}(v_7) = \text{Road}$,

$lab_{LOC}(v_{13}) = \text{Pasture}$, $lab_{LOC}(v_{14}) = \text{Village}$, $lab_{LOC}(v_{16}) = \text{Inn}$

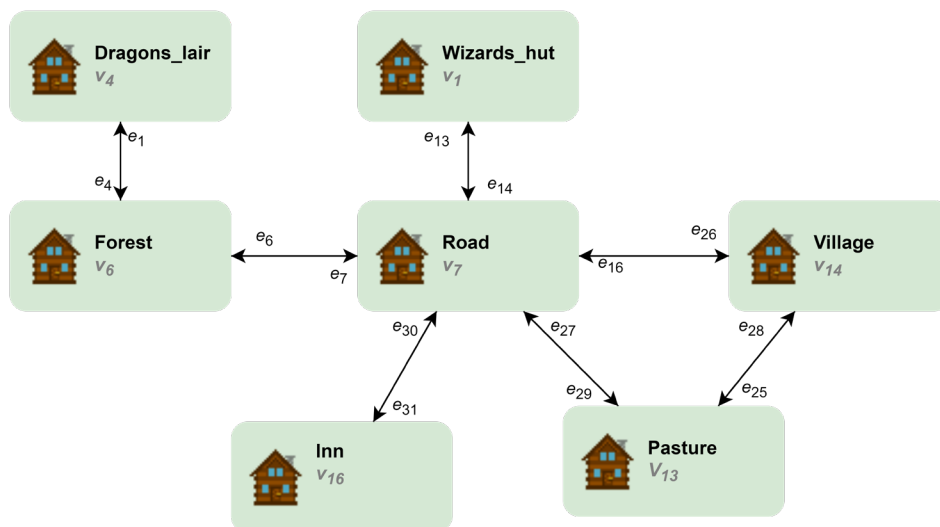
$s_{LOC}(e_1) = v_6$, $t_{LOC}(e_1) = v_4$, $s_{LOC}(e_4) = v_4$, $t_{LOC}(e_4) = v_6$, $s_{LOC}(e_6) = v_7$, $t_{LOC}(e_6) = v_6$,

$s_{LOC}(e_7) = v_6$, $t_{LOC}(e_7) = v_7$, $s_{LOC}(e_{13}) = v_7$, $t_{LOC}(e_{13}) = v_1$, $s_{LOC}(e_{14}) = v_1$, $t_{LOC}(e_{14}) = v_7$,

$s_{LOC}(e_{16}) = v_{14}$, $t_{LOC}(e_{16}) = v_7$, $s_{LOC}(e_{25}) = v_{14}$, $t_{LOC}(e_{25}) = v_{13}$, $s_{LOC}(e_{26}) = v_7$, $t_{LOC}(e_{26}) = v_{14}$,

$s_{LOC}(e_{27}) = v_{13}$, $t_{LOC}(e_{27}) = v_7$, $s_{LOC}(e_{28}) = v_{13}$, $t_{LOC}(e_{28}) = v_{14}$, $s_{LOC}(e_{29}) = v_7$, $t_{LOC}(e_{29}) = v_{13}$,

$s_{LOC}(e_{30}) = v_{16}$, $t_{LOC}(e_{30}) = v_7$, $s_{LOC}(e_{31}) = v_7$, $t_{LOC}(e_{31}) = v_{16}$



Rys. 7. Graf lokacji.

Graf postaci

Niech L_{CHAR} będzie alfabetem nazw postaci.

Grafem postaci będzie system $CHAR = (V_{CHAR}, E_{CHAR}, s_{CHAR}, t_{CHAR}, lab_{CHAR})$, gdzie:

V_{CHAR} – wierzchołki reprezentujące postaci,

E_{CHAR} – krawędzie wskazujące zależności między postaciami za pomocą funkcji s_{CHAR}, t_{CHAR} ,

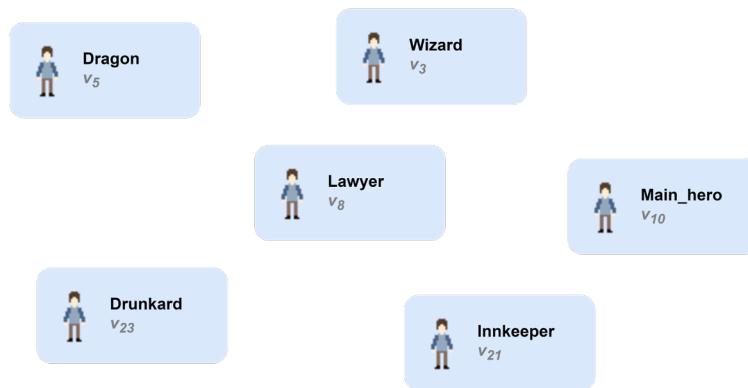
lab_{CHAR} – funkcja przypisująca nazwy postaci wierzchołkom.

$L_{CHAR} = \{\text{Dragon, Drunkard, Innkeeper, Lawyer, Main_hero, Wizard}\}$

$V_{CHAR} = \{v_3, v_5, v_8, v_{10}, v_{21}, v_{23}\}, E_{CHAR} = \emptyset$

$lab_{CHAR}(v_3) = \text{Wizard}, lab_{CHAR}(v_5) = \text{Dragon}, lab_{CHAR}(v_8) = \text{Lawyer},$

$lab_{CHAR}(v_{10}) = \text{Main_hero}, lab_{CHAR}(v_{21}) = \text{Innkeeper}, lab_{CHAR}(v_{23}) = \text{Drunkard}$



Rys. 8. Graf postaci.

Graf przedmiotów

Niech L_{ITEM} będzie alfabetem nazw przedmiotów.

Grafem przedmiotów będzie system $ITEM = (V_{ITEM}, E_{ITEM}, s_{ITEM}, t_{ITEM}, lab_{ITEM})$, gdzie:

V_{ITEM} – wierzchołki reprezentują przedmioty,

E_{ITEM} – krawędzie wskazujące zawieranie przedmiotów za pomocą funkcji s_{CHAR}, t_{CHAR} ,

lab_{ITEM} – funkcja przypisująca nazwy przedmiotów wierzchołkom.

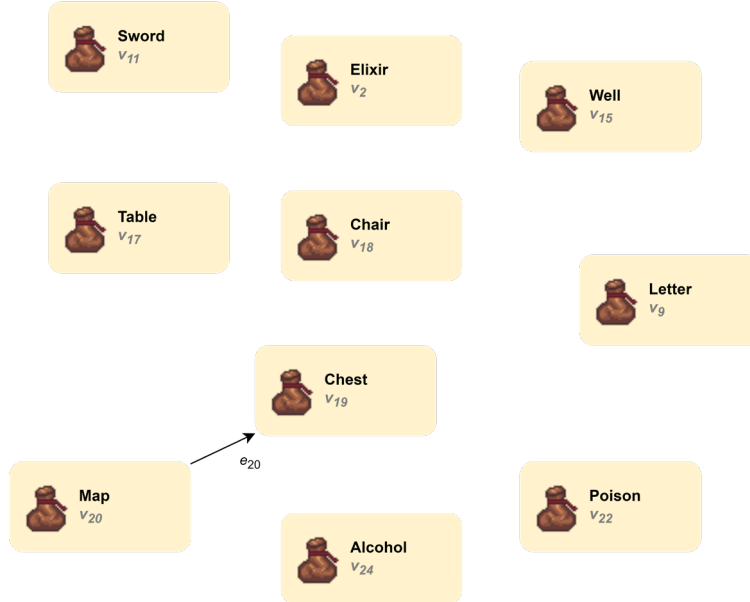
$L_{ITEM} = \{\text{Alcohol, Chair, Chest, Elixir, Letter, Map, Poison, Sword, Table, Well}\}$

$V_{ITEM} = \{v_2, v_9, v_{11}, v_{15}, v_{17}, v_{18}, v_{19}, v_{20}, v_{22}, v_{24}\}$

$E_{ITEM} = \{e_{20}\}$

$lab_{ITEM}(v_2) = \text{Elixir}$, $lab_{ITEM}(v_9) = \text{Letter}$, $lab_{ITEM}(v_{11}) = \text{Sword}$, $lab_{ITEM}(v_{15}) = \text{Well}$,
 $lab_{ITEM}(v_{17}) = \text{Table}$, $lab_{ITEM}(v_{18}) = \text{Chair}$, $lab_{ITEM}(v_{19}) = \text{Chest}$, $lab_{ITEM}(v_{20}) = \text{Map}$,
 $lab_{ITEM}(v_{22}) = \text{Poison}$, $lab_{ITEM}(v_{24}) = \text{Alcohol}$

$SITEM(e_{20}) = v_{20}$, $tITEM(e_{20}) = v_{19}$



Rys. 9. Graf przedmiotów.

Graf narracji

Niech L_{PLOT} będzie alfabetem nazw informacji fabularnych.

Grafem informacji będzie system $PLOT = (V_{PLOT}, E_{PLOT}, S_{PLOT}, t_{PLOT}, lab_{PLOT})$, gdzie:

V_{PLOT} – wierzchołki reprezentujące elementy fabularne,

E_{PLOT} – krawędzie relacji między informacjami, jest to zbiór pusty a funkcje są pusto spełnione,

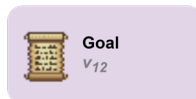
lab_{PLOT} – funkcja przypisująca nazwy informacji wierzchołkom.

$L_{PLOT} = \{\text{Goal}\}$

$V_{PLOT} = \{v_{12}\}$

$E_{PLOT} = \emptyset$

$lab_{PLOT}(v_{12}) = \text{Goal}$



Rys. 10. Graf narracji.

Powyżej zdefiniowany graf uzupełniamy o atrybuty.

Niech A będzie zbiorem atrybutów. Atrybut a jest funkcją określoną na zbiorze obiektów.

Definicja 2 – Atrybutowany etykietowany graf skierowany

Atrybutowanym etykietowanym grafem skierowanym jest system:

$$G = (C, attr), \quad (2)$$

gdzie:

- $C = (V, E, s, t, lab)$ jest etykietowanym grafem skierowanym (por. def. 1),
- $attr : V \cup E \rightarrow 2^A$ jest funkcją atrybutowania.

Przykład 2

Zbudujmy cztery nowe grafy atrybutowane LOC , $CHAR$, $ITEM$, $PLOT$ w oparciu o ich odpowiedniki z *Przykładu 1*, definiując atrybuty węzłów przedstawionych tam grafów.

a_{hp} , $a_{money} \in A$ określają siłę życiową i siłę nabywczą bohatera: dziedziną funkcji jest V_{CHAR} a dziedziną wartości $D_{a_{hp}}$, $D_{a_{money}}$ jest zbiór liczb rzeczywistych.

a_{isDead} , $a_{isWounded} \in A$ mają wartości logiczne: *true*, *false* i określają stan bohatera: martwy, ranny. Dziedziną funkcji jest V_{CHAR} .

$a_{value} \in A$ określa wartość przedmiotu. Dziedziną funkcji jest V_{ITEM} a dziedziną wartości tego atrybutu jest zbiór liczb rzeczywistych dodatnich.

$a_{knowledge} \in A$ określa zakres wiedzy o świecie. Dziedziną funkcji jest V_{PLOT} . Jest to atrybut tekstowy. Oprócz tego definiujemy atrybuty dodatkowe zdefiniowane tylko dla wierzchołków o danej etykietce (np. $\forall v$ takiego, że $lab(v) = „Bandit”$) lub dla konkretnego wskazanego wierzchołka.

Niech A będzie zbiorem atrybutów zdefiniowanym jako:

$$A = \{a_{hp}, a_{money}, a_{isDead}, a_{isWounded}, a_{value}, a_{knowledge}, a_{isWizardsProperty}, a_{motto}, a_{isTroublemaker}, a_{height}\}$$

Zdefiniujmy funkcje atrybutowania w następujący sposób:

$$attr(v_2) = \{a_{value}, a_{isWizardsProperty}\}, \quad attr(v_3) = \{a_{hp}, a_{money}, a_{isDead}, a_{isWounded}, a_{height}\}$$

$$attr(v_5) = \{a_{hp}, a_{money}, a_{isDead}, a_{isWounded}, a_{height}\}, \quad attr(v_8) = \{a_{hp}, a_{money}, a_{isDead}, a_{isWounded}, a_{motto}\}$$

$$attr(v_9) = \{a_{value}\}, \quad attr(v_{10}) = \{a_{hp}, a_{money}, a_{isDead}, a_{isWounded}, a_{motto}\}, \quad attr(v_{11}) = \{a_{value}\}$$

$$attr(v_{12}) = \{a_{knowledge}\}, \quad attr(v_{15}) = \{a_{value}\}, \quad attr(v_{17}) = \{a_{value}\}, \quad attr(v_{18}) = \{a_{value}\}$$

$$attr(v_{19}) = \{a_{value}\}, \quad attr(v_{20}) = \{a_{hp}, a_{money}, a_{isDead}, a_{isWounded}, a_{motto}\}, \quad attr(v_{21}) = \{a_{value}\}$$

$$attr(v_{22}) = \{a_{isTroublemaker}, a_{isDead}, a_{money}, a_{motto}\}, \quad attr(v_{23}) = \{a_{value}\}$$

$$attr(v_x) = \{\} \text{ dla każdego } v_x \in V_{LOC}$$

$$attr(e_x) = \{\} \text{ dla każdego } e_x \in E_{LOC} \cup E_{CHAR} \cup E_{ITEM} \cup E_{PLOT}$$

Definicja 3 – Instancja grafu atrybutowanego

Instancją grafu atrybutowanego jest para:

$$G_I = (G, val), \quad (3)$$

gdzie:

- $G = (C, attr)$ jest etykietowanym skierowanym grafem atrybutowanym (por. def. 2),
- $val : (V \cup E) \times A \rightarrow D$, gdzie $D = \{D_a\}_{a \in A}$, jest częściową funkcją, taką że dla wszystkich $x \in V \cup E$ oraz $a \in A$ jeśli $a \in attr(x)$ wtedy $val(x, a) \in D_a$.

Przykład 3

Tworzymy cztery instancje grafów z *Przykładu 2* wyznaczając wartości funkcji val w następujący sposób.

$val(v_2, a_{value}) = 10$, $val(v_2, a_{isWizardsProperty}) = true$,
 $val(v_3, a_{hp}) = 100$, $val(v_3, a_{money}) = 100$, $val(v_3, a_{isDead}) = false$, $val(v_3, a_{isWounded}) = false$,
 $val(v_5, a_{hp}) = 100$, $val(v_5, a_{money}) = 0$, $val(v_5, a_{isDead}) = false$, $val(v_5, a_{isWounded}) = false$,
 $val(v_8, a_{hp}) = 100$, $val(v_8, a_{money}) = 100$, $val(v_8, a_{isDead}) = false$, $val(v_8, a_{isWounded}) = false$,
 $val(v_8, a_{motto}) = Ignorantia iuris nocet$,
 $val(v_9, a_{value}) = 10$,
 $val(v_{10}, a_{hp}) = 100$, $val(v_{10}, a_{money}) = 10$, $val(v_{10}, a_{isDead}) = false$, $val(v_{10}, a_{isWounded}) = false$,
 $val(v_{10}, a_{motto}) = Homo sum, humani nihil a me alienum puto$,
 $val(v_{11}, a_{value}) = 100$,
 $val(v_{12}, a_{knowledge}) = Celem gry jest dotarcie do zamorskiej krainy$,
 $val(v_{15}, a_{value}) = 1000$,
 $val(v_{17}, a_{value}) = 1$,
 $val(v_{18}, a_{value}) = 1000$,
 $val(v_{19}, a_{value}) = 10$,
 $val(v_{20}, a_{hp}) = 100$, $val(v_{20}, a_{money}) = 100$, $val(v_{20}, a_{isDead}) = false$, $val(v_{20}, a_{isWounded}) = false$,
 $val(v_{20}, a_{motto}) = In vino veritas$,
 $val(v_{21}, a_{value}) = 10$,
 $val(v_{22}, a_{isTroublemaker}) = true$, $val(v_{22}, a_{isDead}) = false$, $val(v_{22}, a_{money}) = 0$,
 $val(v_{22}, a_{motto}) = Spiritus flat ubi vult$,
 $val(v_{23}, a_{value}) = 1$

W następnym kroku definiujemy graf warstwowy.

Niech $\Gamma(L, A)$ będzie zbiorem wszystkich etykietowanych skierowanych atrybutowanych grafów $G = (V_G, E_G, s_G, t_G, lab_G, attr_G)$, z etykietami ze zbioru L i atrybutami ze zbioru A .

Niech $\Gamma_I(L, A)$ będzie zbiorem instancji grafów z $\Gamma(L, A)$.

Niech L_H będzie zbiorem takim, że $L \cap L_H = \emptyset$. L_H będziemy nazywać alfabetem etykiet warstwowych.

Definicja 4 – Graf n -warstwowy

Grafem n -warstwowym dla liczby naturalnej $n \geq 2$ nazywamy:

$$H = (X, ch, E, s, t, lab), \quad (4)$$

gdzie:

- X jest n -elementowym zbiorem wierzchołków zwanych warstwami, takim że $\forall G \in \Gamma(L, A) : V_G \cap X = \emptyset$,
- $ch : X \ni x \rightarrow G_{ch(x)} \in \Gamma(L, A)$ jest funkcją przypisującą każdej warstwie zagnieżdżony w niej graf tak, że dla wszystkich grafów $G_{ch(x)}$, $x \in X$ ich zbiory wierzchołków, czyli $V_{ch(x)}$, są rozłączne, zbiory krawędzi, czyli $E_{ch(x)}$, są rozłączne oraz zbiory użytych etykiet, czyli $lab_{ch(x)}(V_{ch(x)})$, są rozłączne,
- E jest zbiorem krawędzi zwanych krawędziami zewnętrznymi, rozłącznym ze wszystkimi zbiorami krawędzi $E_{ch(x)}$ dla $x \in X$,
- $s, t : E \rightarrow \bigcup_{x \in X} V_{ch(x)}$ są odpowiednio funkcją źródłową i docelową dla krawędzi zewnętrznych,
- krawędzie zewnętrzne są międzywarstwowe, tzn. $\forall x \in X, e \in E : s(e) \in V_{ch(x)} \Rightarrow t(e) \notin V_{ch(x)}$,
- $lab : X \rightarrow L_H$ jest iniektywną funkcją etykietowania warstw.

Definicja 5 – Instancja grafu n -warstwowego

Instancją grafu warstwowego nazywamy graf warstwowy, w którym dla wszystkich warstw $x \in X$ grafy $G_{ch(x)}$ zostały zastąpione ich instancjami.

Przykład 4

Instancji grafu warstwowego używamy do opisu stanu świata gry, stwórzmy więc przykładowy stan świata gry, czyli GWSG. Możemy zbudować stan świata gry, którego warstwami będą instancje z *Przykładu 3*, ponieważ mają one rozłączne alfabety etykiet, zbiory wierzchołków i rozłączne zbiory krawędzi.

Wyznaczmy więc graf warstwowy dla $n = 4$.

$L = \{\text{Dragons_lair, Forest, Inn, Pasture, Road, Village, Wizards_hut, Dragon, Drunkard, Innkeeper, Lawyer, Main_hero, Wizard, Alcohol, Chair, Chest, Elixir, Letter, Map, Poison, Sword, Table, Well, Goal}\}$

$A = \{a_{hp}, a_{money}, a_{isDead}, a_{isWounded}, a_{value}, a_{knowledge}, a_{isWizardsProperty}, a_{motto}, a_{isTroublemaker}, a_{height}\}$

$L_H = \{\text{Locations, Characters, Items, Plot_elements}\}$

$X = \{x_1, x_2, x_3, x_4\}$

$E = \{e_2, e_3, e_5, e_8, e_9, e_{10}, e_{11}, e_{12}, e_{15}, e_{17}, e_{18}, e_{19}, e_{21}, e_{22}, e_{23}, e_{24}\}$

$ch(x_1) = LOC, ch(x_2) = CHAR, ch(x_3) = ITEM, ch(x_4) = PLOT$

$s(e_2) = v_2 \in V_{ITEM}, t(e_2) = v_1 \in V_{LOC}, V_{ITEM} \cap V_{LOC} = \emptyset$

$s(e_3) = v_3 \in V_{CHAR}, t(e_3) = v_1 \in V_{LOC}, V_{CHAR} \cap V_{LOC} = \emptyset$

$s(e_5) = v_5 \in V_{CHAR}, t(e_5) = v_4 \in V_{LOC},$

$s(e_8) = v_8 \in V_{CHAR}, t(e_8) = v_7 \in V_{LOC},$

$s(e_9) = v_9 \in V_{ITEM}, t(e_9) = v_8 \in V_{CHAR},$

$s(e_{10}) = v_{10} \in V_{CHAR}, t(e_{10}) = v_7 \in V_{LOC},$

$s(e_{11}) = v_{11} \in V_{ITEM}, t(e_{11}) = v_{10} \in V_{CHAR},$

$s(e_{12}) = v_{12} \in V_{PLOT}, t(e_{12}) = v_{10} \in V_{CHAR}, V_{PLOT} \cap V_{CHAR} = \emptyset$

$s(e_{15}) = v_{15} \in V_{ITEM}, t(e_{15}) = v_{14} \in V_{LOC},$

$s(e_{17}) = v_{17} \in V_{ITEM}, t(e_{17}) = v_{16} \in V_{LOC},$

$s(e_{18}) = v_{18} \in V_{ITEM}, t(e_{18}) = v_{16} \in V_{LOC},$

$s(e_{19}) = v_{19} \in V_{ITEM}, t(e_{19}) = v_{16} \in V_{LOC},$

$s(e_{21}) = v_{21} \in V_{CHAR}, t(e_{21}) = v_{16} \in V_{LOC},$

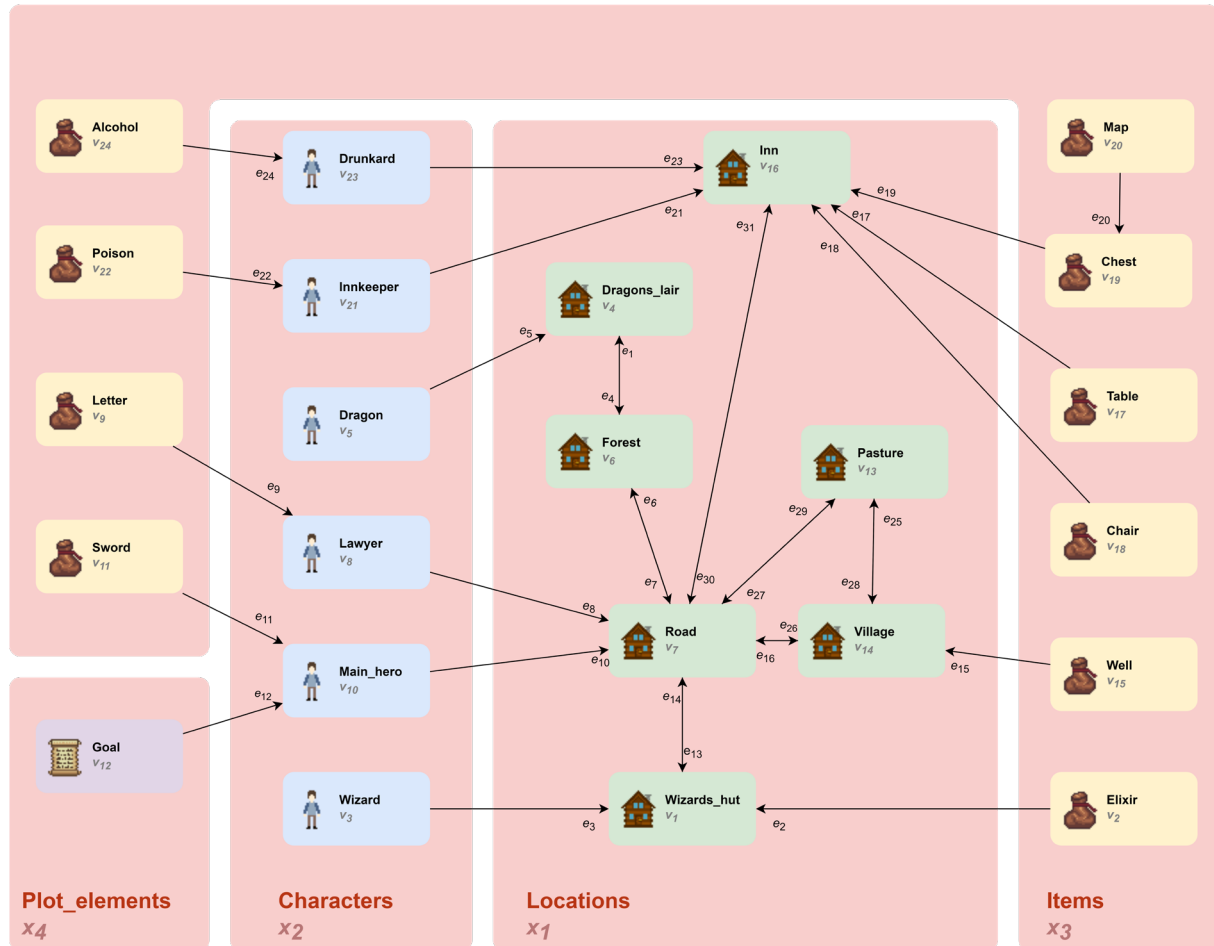
$s(e_{22}) = v_{22} \in V_{ITEM}, t(e_{22}) = v_{21} \in V_{CHAR},$

$s(e_{23}) = v_{23} \in V_{CHAR}, t(e_{23}) = v_{16} \in V_{LOC},$

$s(e_{24}) = v_{24} \in V_{ITEM}, t(e_{24}) = v_{23} \in V_{CHAR},$

$lab(x_1) = \text{Locations}, lab(x_2) = \text{Characters}, lab(x_3) = \text{Items}, lab(x_4) = \text{Plot_elements}$

Tak zbudowana instancja jest już pełnoprawnym opisem stanu świata gry w rozumieniu modelu przedstawionego w niniejszej rozprawie. Symbolicznie przedstawiona jest na rysunku 11.



Rys. 11. Wizualizacja GWSG zbudowanego w *Przykładzie 4*. Atrybuty wierzchołków i ich wartości nie zostały umieszczone na rysunku, aby nie zmniejszać jego czytelności.

Na potrzeby łatwego operowania na grafach wprowadzimy definicje elementów, które ułatwią nam dokonywanie transformacji grafowych.

Definicja 6 – Podgraf i graf częściowy

Dla grafu G graf G' jest podgrafem, jeśli zbiór wierzchołków grafu G' jest podzbiorem zbioru wierzchołków grafu G oraz zbiór krawędzi grafu G' zawiera wszystkie krawędzie, których wierzchołki źródłowy i docelowy są elementami zbioru wierzchołków grafu G' .

Graf niespełniający tego warunku dla krawędzi jest grafem częściowym.

Niech m będzie liczbą naturalną.

Definicja 7 – Ścieżka w grafie

Ścieżką o długości m w n -warstwowym grafie H nazwiemy ciąg m krawędzi $e_1, \dots, e_m$ takich, że:

jeżeli $1 \leq i < m$ to $t_P(e_i) = s_P(e_{i+1})$

gdzie:

- $s_P(e) = \begin{cases} s_H(e), & \text{jeśli } e \in E_H \\ s_{ch(x)}(e), & \text{jeśli } e \in E_{ch(x)}, \text{ dla } x \in X_H \end{cases}$ to jednolita funkcja źródłowa krawędzi,
- $t_P(e) = \begin{cases} t_H(e), & \text{jeśli } e \in E_H \\ t_{ch(x)}(e), & \text{jeśli } e \in E_{ch(x)} \text{ dla } x \in X_H \end{cases}$ to jednolita funkcja docelowa krawędzi.

Graf jest grafem wielosnopkowym, jeżeli możemy wyróżnić tylko jedną warstwę, w której elementy łączą się ze sobą w dowolny sposób (np. w cykle) a elementy z pozostałych warstw łączą się w drzewa, których korzeniami są wierzchołki z warstwy wyróżnionej.

Definicja 8 – Graf wielosnopkowy

Graf n -warstwowy $H = (X, ch, E, s, t, lab)$ będziemy nazywać wielosnopkowym, jeżeli istnieje warstwa $x_R \in X$ taka, że:

- nie istnieje krawędź zewnętrzna $e \in E$ taka, że jej wierzchołek źródłowy $s(e)$ należy do grafu $G_{ch(x_R)}$ zagnieżdżonego w wyróżnionej warstwie x_R , a wierzchołek docelowy $t(e)$ nie należy do tego grafu,
- w grafie nie ma ścieżek, które przechodzą dwa razy przez ten sam wierzchołek nienależący do zbioru wierzchołków grafu $G_{ch(x_R)}$,
- z każdego wierzchołka nienależącego do zbioru wierzchołków grafu $G_{ch(x_R)}$ wychodzi dokładnie jedna krawędź wewnętrzna lub zewnętrzna.

Definicja 9 – Podgraf snopkowy

Podgrafem snopkowym nazywamy podgraf grafu wielosnopkowego indukowany wszystkimi ścieżkami kończącymi się w danym wierzchołku z warstwy x_R i nie zawierającymi innych wierzchołków z warstwy x_R .

Definicja 10 – Podgraf półsnopkowy

Podgraf półsnopkowy to podgraf grafu wielosnopkowego indukowany wszystkimi ścieżkami kończącymi się w danym wierzchołku nie należącym do warstwy x_R .

Podgrafy snopkowe będą pomocne przy określaniu kontekstu potrzebnego do transformacji grafowej a podgrafy półsnopkowe przy bezpośrednim wskazywaniu zakresu modyfikacji grafu podczas transformacji.

Przykład 5

Możemy w grafie zdefiniowanym w *Przykładzie 4* wyróżnić siedem podgrafów snopkowych, z czego tylko pięć nietrywialnych, tzn. zawierających więcej niż jeden węzeł.

Snopek zbudowany na lokacji v_1 zawiera wierzchołki v_1, v_2, v_3 i następujące ścieżki:

$$v_2 \in V_{ch(x_3)} \rightarrow v_1 \in V_{ch(x_1)}$$

$$v_3 \in V_{ch(x_2)} \rightarrow v_1 \in V_{ch(x_1)}$$

Snopek zbudowany na lokacji v_4 zawiera wierzchołki v_4, v_5 i następującą ścieżkę:

$$v_5 \in V_{ch(x_2)} \rightarrow v_4 \in V_{ch(x_1)}$$

Snopek zbudowany na lokacji v_7 zawiera wierzchołki $v_7, v_8, v_9, v_{10}, v_{11}, v_{12}$ i następujące ścieżki:

$$v_9 \in V_{ch(x_3)} \rightarrow v_8 \in V_{ch(x_2)} \rightarrow v_7 \in V_{ch(x_1)}$$

$$v_{11} \in V_{ch(x_3)} \rightarrow v_{10} \in V_{ch(x_2)} \rightarrow v_7 \in V_{ch(x_1)}$$

$$v_{12} \in V_{ch(x_4)} \rightarrow v_{10} \in V_{ch(x_2)} \rightarrow v_7 \in V_{ch(x_1)}$$

Snopek zbudowany na lokacji v_{14} zawiera wierzchołki v_{14}, v_{15} i następującą ścieżkę:

$$v_{15} \in V_{ch(x_3)} \rightarrow v_{14} \in V_{ch(x_1)}$$

Snopek zbudowany na lokacji v_{16} zawiera wierzchołki $v_{16}, v_{17}, v_{18}, v_{19}, v_{20}, v_{21}, v_{22}, v_{23}, v_{24}$ i następujące ścieżki:

$$v_{17} \in V_{ch(x_3)} \rightarrow v_{16} \in V_{ch(x_1)}$$

$$v_{18} \in V_{ch(x_3)} \rightarrow v_{16} \in V_{ch(x_1)}$$

$$v_{20} \in V_{ch(x_3)} \rightarrow v_{19} \in V_{ch(x_3)} \rightarrow v_{16} \in V_{ch(x_1)}$$

$$v_{22} \in V_{ch(x_3)} \rightarrow v_{21} \in V_{ch(x_2)} \rightarrow v_{16} \in V_{ch(x_1)}$$

$$v_{24} \in V_{ch(x_3)} \rightarrow v_{23} \in V_{ch(x_2)} \rightarrow v_{16} \in V_{ch(x_1)}$$

Graf zawiera także siedemnaście podgrafów półsnopkowych, z tego pięć nietrywialnych.

Półsnoppek zbudowany na wierzchołku v_8 zawiera wierzchołki v_8, v_9 i następującą ścieżkę:

$$v_9 \in V_{ch(x_3)} \rightarrow v_8 \in V_{ch(x_2)}$$

Półsnoppek zbudowany na wierzchołku v_{10} zawiera wierzchołki v_{10}, v_{11}, v_{12} i następujące ścieżki:

$$v_{11} \in V_{ch(x_3)} \rightarrow v_{10} \in V_{ch(x_2)}$$

$$v_{12} \in V_{ch(x_4)} \rightarrow v_{10} \in V_{ch(x_2)}$$

Półsnoppek zbudowany na wierzchołku v_{19} zawiera wierzchołki v_{19}, v_{20} i następującą ścieżkę:

$$v_{20} \in V_{ch(x_3)} \rightarrow v_{19} \in V_{ch(x_2)}$$

Półsnoppek zbudowany na wierzchołku v_{21} zawiera wierzchołki v_{21}, v_{22} i następującą ścieżkę:

$$v_{22} \in V_{ch(x_3)} \rightarrow v_{21} \in V_{ch(x_2)}$$

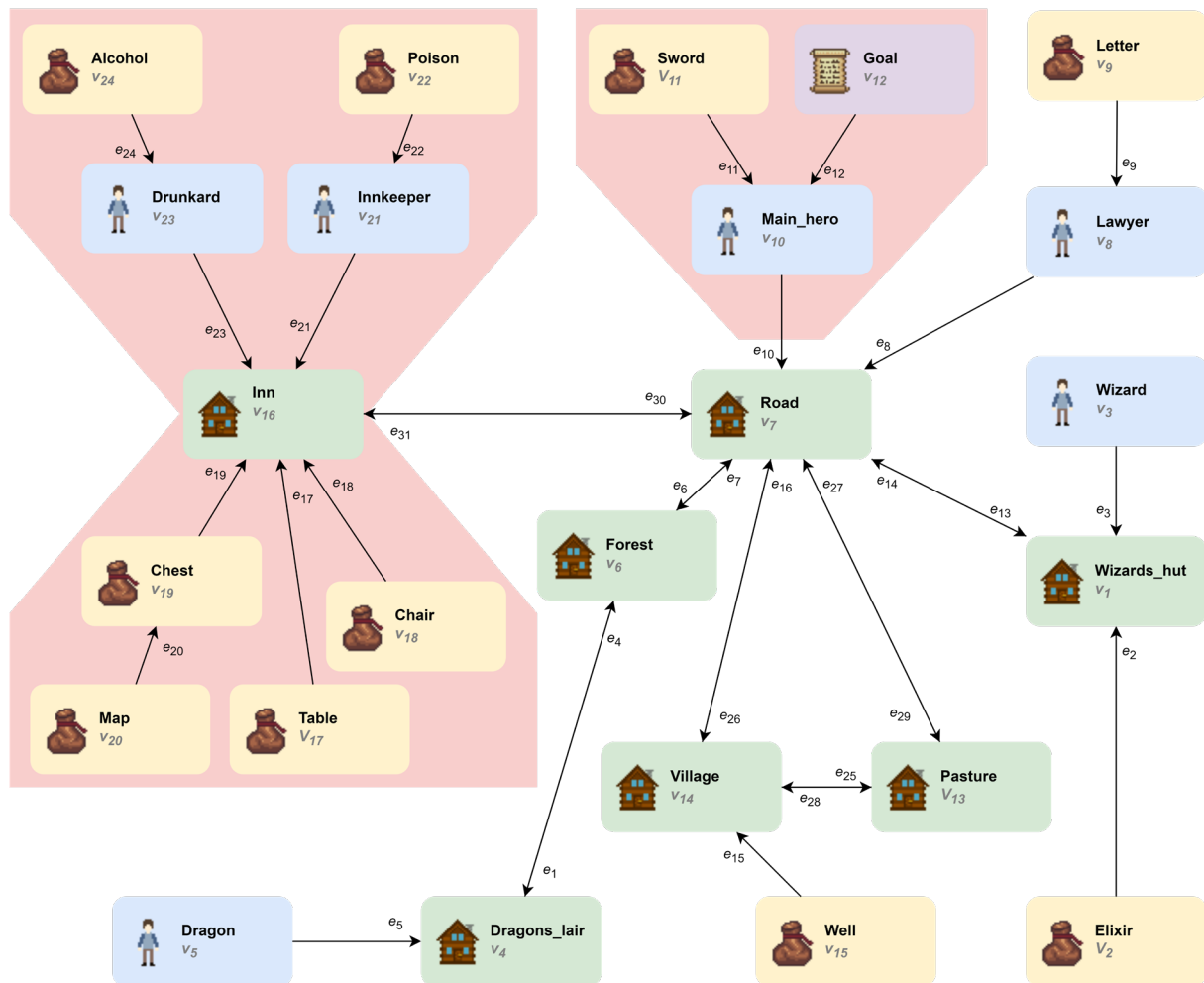
Półsnoppek zbudowany na wierzchołku v_{23} zawiera wierzchołki v_{23}, v_{24} i następującą ścieżkę:

$$v_{24} \in V_{ch(x_3)} \rightarrow v_{23} \in V_{ch(x_2)}$$

Na rysunku 12 przedstawiony został GWSG zdefiniowany w *Przykładzie 4*, tym razem jednak zrezygnowano z rysowania wierzchołków warstw, wykorzystując czerwone tło do wyróżnienia przykładowego podgrafu snopkowego i półsnopkowego. Przynależność wierzchołków do poszczególnych warstw pozostaje oznaczona kolorami i ikonami. Kolor zielony oznacza węzły zagnieżdżone w x_1 , kolor niebieski w x_2 , żółty w x_3 a fioletowy w x_4 .

Najbardziej rozbudowany podgraf snopkowy, składający się z dziewięciu węzłów, został zaznaczony na rysunku czerwonym tłem przypominającym snopek (po lewej stronie).

Najbardziej rozbudowany podgraf półsnopkowy, składający się z trzech węzłów, został przedstawiony na rysunku czerwonym tłem, które można uznać za kształt odpowiadający górnej połowie snopka. Znajduje się on na prawo od podgrafu snopkowego.



Rys. 12. GWSG z zaznaczonym podgrafem snopkowym i półsnopkowym.

Wprowadźmy pojęcie częściowego etykietowania.

Definicja 11 – Graf atrybutowany z częściowym etykietowaniem

K jest grafem atrybutowanym z częściowym etykietowaniem, jeśli K jest zdefiniowane tak jak G w Definicji 2, z warunkiem, że funkcja $lab : V \rightarrow L$ jest funkcją częściową.

Zbiór argumentów funkcji częściowej lab będziemy oznaczać przez $dom(lab)$. W kontekście grafów z pełnym etykietowaniem (por. Definicja 2) $dom(lab)$ będzie oznaczało V .

Definicja 12 – Instancja grafu atrybutowanego z częściowym etykietowaniem

$K_I = (K, val)$ jest instancją grafu z częściowym etykietowaniem, jeśli K jest grafem atrybutowanym z częściowym etykietowaniem, a val jest zdefiniowane tak jak w Definicji 3.

Niech $\Delta(L, A)$ będzie zbiorem wszystkich grafów atrybutowanych z częściowym bądź pełnym etykietowaniem nad alfabetem etykiet L i zbiorem atrybutów A .

Niech $\Delta_I(L, A)$ będzie zbiorem wszystkich instancji grafów z $\Delta(L, A)$.

Co za tym idzie, $\Gamma(L, A) \subsetneq \Delta(L, A)$ oraz $\Gamma_I(L, A) \subsetneq \Delta_I(L, A)$.

Definicja 13 – Graf n -warstwowy z częściowym bądź pełnym etykietowaniem

M jest grafem n -warstwowym z częściowym bądź pełnym etykietowaniem, jeśli M jest zdefiniowane tak jak H w Definicji 4, ale zamiast funkcji $ch : X \rightarrow \Gamma(L, A)$ jest funkcja $ch : X \rightarrow \Delta(L, A)$.

Definicja 14 – Instancja grafu n -warstwowego z częściowym bądź pełnym etykietowaniem

Instancją grafu n -warstwowego z częściowym bądź pełnym etykietowaniem nazywamy graf n -warstwowy z częściowym bądź pełnym etykietowaniem, w którym dla wszystkich warstw $x \in X$ grafy $G_{ch(x)}$ zostały zastąpione ich instancjami.

Definicja 15 – Dopasowanie generyczne grafów atrybutowanych z częściowym bądź pełnym etykietowaniem oraz ich instancji

Niech K będzie grafem atrybutowanym z częściowym bądź pełnym etykietowaniem (por. def. 11 i 2). Niech Q będzie grafem atrybutowanym z częściowym bądź pełnym etykietowaniem lub instancją takiego grafu (por. def. 11 i 2 lub def. 12 i 3). Dopasowanie generyczne $k : K \rightarrow Q$ jest parą iniektywnych funkcji $k_E : E_K \rightarrow E_Q$ oraz $k_V : V_K \rightarrow V_Q$ zachowujących własności wierzchołków i krawędzi:

- $\forall e \in E_K : k_V(s_K(e)) = s_Q(k_E(e))$
- $\forall e \in E_K : k_V(t_K(e)) = t_Q(k_E(e))$
- $\forall v \in V_K : v \in \text{dom}(\text{lab}_K) \implies \text{lab}_K(v) = \text{lab}_Q(k_V(v))$
- $\forall v \in V_K : \text{attr}_K(v) \subseteq \text{attr}_Q(k_V(v))$
- $\forall e \in E_K : \text{attr}_K(e) \subseteq \text{attr}_Q(k_E(e))$

Definicja 16 – Dopasowanie generyczne grafów n -warstwowych z częściowym bądź pełnym etykietowaniem oraz ich instancji

Niech M będzie grafem n -warstwowym z częściowym bądź pełnym etykietowaniem (por. def. 13 i 4). Niech N będzie grafem n -warstwowym z częściowym bądź pełnym etykietowaniem lub instancją takiego grafu (por. def. 13 i 4 lub 14 i 5). Dopasowanie generyczne $m : M \rightarrow N$ jest trójką złożoną z funkcji $m_X : X_M \rightarrow X_N$, funkcji $m_E : E_M \rightarrow E_N$ i rodziny dopasowań generycznych $\{k_x\}_{x \in X_M}$.

Dla dopasowania generycznego k_x będącego parą funkcji, pierwszą funkcję z tej pary oznaczamy: k^1_x . Wtedy definiujemy funkcję $k_V : \bigcup_{x \in X_M} V_{ch_M(x)} \rightarrow \bigcup_{x \in X_N} V_{ch_N(x)}$ jako:

$$k_V(v) = \begin{cases} k^1_x(v) & \text{jeśli } v \in V_{ch_M(x)} \end{cases}$$

Trójka ta ma następujące własności:

- a) m_X jest bijekcją,
- b) $\forall x \in X_M: lab_M(x) = lab_N(m_X(x))$,
- c) $\forall x \in X_M: k_x$ jest dopasowaniem generycznym z grafu $ch_M(x)$ w graf lub instancję $ch_N(m_X(x))$ (por. def. 15),
- d) $\forall e \in E_M: k_V(s_M(e)) = s_N(m_E(e))$,
- e) $\forall e \in E_M: k_V(t_M(e)) = t_N(m_E(e))$,

Dopasowania istnieją tylko pomiędzy grafami mającymi taką samą liczbę tak samo etykietowanych warstw. Dopasowania z definicji są iniektywne.

Modyfikacji grafu będziemy dokonywać za pomocą struktury zwanej produkcją grafową. Grafowa procedura generacyjna jest systemem transformowania grafów według formalnych reguł zwanych produkcjami.

Klasyczne systemy generacyjne transformacji grafowych mają produkcje składające się z dwóch grafów, zwanych lewą i prawą stroną produkcji. Produkcja może być użyta do przetransformowania danego grafu tylko wtedy, jeśli uda się znaleźć monomorfizm (iniektyny homomorfizm) odwzorowujący lewą stronę produkcji w przekształcany graf. Zaaplikowanie transformacji polega w tych systemach na usunięciu z grafu fragmentu dopasowanego do lewej strony i wstawienie na jego miejsce kopii prawej strony.

W modelu grafowym proponowanym w niniejszej pracy zrezygnowano z prawej strony i podejścia usuń-wstaw, jako zbyt mało elastycznego w praktycznych zastosowaniach. Produkcje

zamiast prawych stron będą miały procedury, które algorytmicznie modyfikują przekształcany graf. Wykonanie ciągu instrukcji zastępuje proces usuwania starego podgrafu, umieszczania nowego i osadzania go.

Prawą stronę produkcji możemy odtwarzać na podstawie instrukcji, ale odtworzenie to będzie pełne tylko dla pewnego podzbioru instrukcji. Zarówno pełne jak i niepełne odtworzenie możemy prezentować w postaci poglądowych wizualizacji (por. rozdz. 5.4, podrozdział *Wizualizacja produkcji*), ale w drugim przypadku prawa strona nie mogłaby stać się podstawą dla wykonania produkcji.

Definicja 17 – Produkcja generyczna

Produkcja generyczna p jest trójką (M, C, P) , gdzie M jest grafem n -warstwowym z częściowym bądź pełnym etykietowaniem, C jest procedurą sprawdzania predykatów stosowalności, a P jest procedurą modyfikującą przekształcany graf.

Definicja 18 – Wyprowadzenie generyczne

Niech N_1 będzie instancją grafu n -warstwowego. Niech $p = (M, C, P)$ będzie produkcją generyczną. Produkcja p jest stosowalna do N_1 , jeżeli istnieje dopasowanie generyczne $m : M \rightarrow N_1$ i jeżeli procedura C , wywołana z argumentami m i N_1 , zwróci wynik oznaczający spełnienie wszystkich sprawdzanych przez nią warunków.

Efektem działania procedury P z argumentami m i N_1 jest nowa instancja N_2 .

Mówimy, że N_2 została wyprowadzona z N_1 za pomocą p .

Relację tę oznaczamy jako: $N_1 \xRightarrow{p} N_2$.

Definicja 19 – System przepisywania

System przepisywania składa się ze startowej instancji n -warstwowego grafu oraz skończonego zbioru produkcji generycznych.

Wyprowadzenie N_2 z N_1 przy pomocy dowolnej produkcji z tego systemu oznaczamy jako:

$N_1 \Rightarrow N_2$. Zwrotne i przechodnie domknięcie tej relacji będziemy oznaczać $\Rightarrow^*$.

4 Założenia systemu StoryGraph

Na podstawie teoretycznego modelu grafowego zaproponowana została specyfikacja standardu implementacji StoryGraph, aktualnie dostępna jako StoryGraph 1.2. Zdefiniowane zostały niezbędne typy danych i wytyczne dotyczące ich szczegółowej implementacji. Standard implementacji jest elementem pośrednim między formalizmem matematycznym a kodem w konkretnym języku programowania. W oparciu o tę specyfikację działa zaimplementowany w języku Python system symulacji procesu decyzyjnego będący podstawą części praktycznej tej pracy i implementacją referencyjną oraz są tworzone dwa inne systemy: system dynamicznych wizualizacji implementowany w silniku Godot i w języku GScript oraz prosta gra fabularna będąca pracą magisterską opartą o silnik zbudowany na pierwszym systemie udostępnionym poprzez sieciowy interfejs programistyczny aplikacji zgodny z zasadami architektury REST (REST API, ang. *representational state transfer application programming interface*), także zaimplementowana w silniku Godot. Między tymi systemami zachodzą różnice na poziomie implementacyjnym, ale każdy z nich spełnia założenia specyfikacji StoryGraph 1.2.

Specyfikacja obejmuje definicję struktury odpowiadającej aktualnemu stanowi świata (GWSG, instancji grafu czterowarstwowego) oraz definicję struktury odpowiadającej produkcji, czyli trójce (M, C, P) , gdzie M jest grafem czterowarstwowym z częściowym bądź pełnym etykietowaniem, C jest procedurą sprawdzania predykatów stosowalności, a P jest procedurą modyfikującą przekształcany graf.

Struktury odpowiadające aktualnemu stanowi świata i lewej stronie produkcji zostały na poziomie specyfikacji uwspólnione, zaproponowano jeden typ danych SHEAF zawierający elementy pozwalające wyrazić oba typy obiektów modelu teoretycznego. Dzięki temu ułatwiony jest proces implementacyjny, w szczególności procedury walidacji mogą opierać się na jednym schemacie JSON Schema, co ułatwia utrzymanie spójności danych.

Procedura sprawdzania predykatów wymaga zdefiniowania typów i sposobu zapisu predykatów oraz wartości zwracanych przez procedurę sprawdzającą. Dodatkowo w predykatkach będziemy odwoływać się do wartości związanych z węzłami dopasowanymi do lewej strony produkcji oraz węzłami z ich pólśnopków, dlatego trzeba zdefiniować typ danych pozwalający na odwołania do węzła.

Procedura modyfikująca opiera się na instrukcjach, których podstawowy zestaw, wymagany w każdej implementacji, także zdefiniujemy. W instrukcjach także odwołujemy się do węzłów dopasowanych do lewej strony produkcji oraz pólśnopków, uspójnimy więc i tę definicję. W instrukcjach wykraczających poza zestaw podstawowy możliwe jest odwołanie się do wierzchołków grafu świata poza snopkami, ale zarówno takie instrukcje jak i sposób odwołania pozostawiamy w gestii programistów poszczególnych implementacji.

Mamy więc zdefiniować:

- Typ danych SHEAF odpowiadający formalizmowi zarówno grafu czterowarstwowego wielosnopkowego z częściowym etykietowaniem jak i jego instancji oraz typy odpowiadające podgrafom pólśnopkowym z korzeniami z różnych warstw.
- Typ danych NODE-MULTIREF pozwalający na odwołanie się do węzła ze snopka dowolnego węzła dopasowanego do lewej strony produkcji.
- Struktury słowników⁶ odpowiadających podstawowym predykatom.
- Struktury słowników odpowiadających podstawowym instrukcjom.

Cechą odróżniającą obiekt JSON opisujący świat od opisującego produkcję jest pusty słownik instrukcji.

Wyjaśnienia notacji

W niniejszym i następnych rozdziałach odwołania do zapisów kodu źródłowego, pseudo-kodu lub zapisu formatu JSON będą oznaczane czcionką Consolas i szarym tłem w przypadku krótkich fragmentów w opisie (przykład: `fragment kodu`) lub wydzielane jako cytat blokowy

⁶ Terminu „słownik” będę używała na określenie struktury danych, która przechowuje pary klucz – wartość, niezależnie od tego, czy będę odnosiła się do języka python (*dictionary*), czy formatu JSON (*JSON object*) czy innych struktur danych (*associative array*).

w przypadku dłuższych przykładów. W pierwszym przypadku fragmenty nie będą wyróżniane dodatkowo cudzysłowami niezależnie od tego, czy w kodzie znajdują się jako łańcuch tekstowy czy jako struktura języka.

Nazwy typów danych specyficznych dla systemu wyróżniane będą kapitalikami. Abstrakcyjne wartości danych: *true*, *false*, *null* zapisywane kursywą. Odwołania do pojęć z formalizmu matematycznego także zaznaczane kursywą, tak jak w definicjach w rozdziale 3.1. Nazwy własne warstw oznaczane cudzysłowem.

4.1 Struktura grafu wielosnopkowego

Typ danych SHEAF, odpowiadający teoretycznej koncepcji grafu wielosnopkowego (por. rozdział 3.1, *Definicja 8*), zdefiniowany jest w oparciu o powszechnie znane typy danych: łańcuchy (ang. *strings*), liczby (ang. *numbers*), wartości logiczne (ang. *boolean*), tablice (ang. *arrays*) i słowniki (ang. *associative arrays*, *dictionaries*). Kluczami słowników zawsze są łańcuchy, natomiast związane z tymi kluczami wartości to *null* lub łańcuchy, liczby, tablice i słowniki. Typ SHEAF reprezentuje graf czterowarstwowy ($n = 4$) o warstwach: „Locations”, „Characters”, „Items” oraz „Narration” lub jego instancję. Warstwą wyróżnioną w rozumieniu definicji grafu wielosnopkowego jest warstwa „Locations”.

Ze względu na specyfikę struktury grafu wielosnopkowego możemy przedstawić go jako tablicę snopków z dodatkową informacją o krawędziach wychodzących z korzenia danego snopka do korzeni innych snopków. SHEAF to słownik z kluczem `Locations` i związaną z tym kluczem tablicą zawierającą wartości typu LOC-DICT. W przypadku grafu pustego tablica jest pusta.

Podgraf snopkowy

LOC-DICT to słownik odpowiadający teoretycznej koncepcji podgrafu snopkowego, który może zawierać:

- Klucz `Name`, z którym związany jest łańcuch będący etykietą wierzchołka. Nie jest wymagany w grafie z częściowym etykietowaniem.
- Klucz `Attributes` i związaną z nim wartość typu ATTR-DICT, czyli słownik o wartościach *null*, tekstowych, logicznych lub liczbowych. Odpowiada wartości funkcji atrybowania (*attr*), jeśli ogranicza się do wartości *null* a funkcji przypisywania wartości (*val*), jeśli jego elementom przypisywane są wartości innych typów.
- Klucze `Characters`, `Items` i `Narration`. Związane z nimi są tablice z wartościami, odpowiednio, typu CHAR-DICT, ITEM-DICT lub NARR-DICT. Te znajdujące się w tablicach słowniki

reprezentują węzły z warstwy odpowiednio postaci, przedmiotów i informacji fabularnych. Wystąpienie węzła w tablicy oznacza istnienie krawędzi, której węzłem docelowym jest wierzchołek zawierający tablicę a źródłowym węzeł w tablicy zawarty. W taki sposób definiowane są krawędzie międzywarstwowe i krawędzie wewnątrzwarstwowe dla warstw innych niż warstwa lokacji.

- Klucz **Connections**. Związana z nim jest tablica z wartościami typu CONN-DICT. Odzworowuje krawędzie wewnątrzwarstwowe w warstwie lokacji, czyli wyróżnionej warstwie grafu wielosnopkowego, wychodzące z danego korzenia snopka. Nie jest to klucz związany z definicją grafu snopkowego jako takiego.
- Klucz **Id**, z którym związany jest łańcuch będący poprawnym identyfikatorem w sensie popularnych języków programowania. Te łańcuchy muszą być unikalne w ramach danego SHEAF. Dla uniknięcia niejednoznaczności powinny być też różne od nazw (etykiet). Klucz ten nie odwzorowuje żadnej własności modelu grafowego, wprowadzony został dla ułatwienia implementacji instrukcji i zapisu struktury w pliku tekstowym.

Podgraf półsnopkowy

CHAR-DICT oraz ITEM-DICT to słowniki odpowiadające koncepcji podgrafu półsnopkowego zbudowane analogicznie jak LOC-DICT, ale nie mogące zawierać klucza **Connections**.

NARR-DICT to słownik taki jak LOC-DICT, ale mogący zawierać co najwyżej klucze **Name**, **Id** i **Attributes**, ponieważ na bazie węzła z warstwy narracyjnej nie można zbudować nietrywialnego podgrafu półsnopkowego. Nie jest to ograniczenie związane z modelem formalnym, zostało narzucone dopiero w specyfikacji systemu.

Reprezentacja krawędzi

Krawędzie wychodzące z węzła w wyróżnionej warstwie lokacji reprezentowane są w słownikach CONN-DICT dołożonych do struktury LOC-DICT definiującej podgraf snopkowy. Złożenie wszystkich słowników CONN-DICT daje nam pełną informację o krawędziach warstwy wyróżnionej. CONN-DICT to słownik zawierający klucz **Destination** i związaną z nim referencję wskazującą na którąś z lokacji w danym SHEAF odpowiadającą wartości funkcji t (funkcji docelowej krawędzi, ang. *target*, por. rozdział 3.1, *Definicja 1*) dla krawędzi wewnętrznych grafu z warstwy lokacji. Może też zawierać klucz **Attributes** i związaną z nim wartość typu ATTR-DICT. Wartości elementów obiektu **Connections** oznaczają inne węzły z warstwy lokacji, z którymi połączona jest dana lokacja. Występowanie takiego elementu w **Connections** oznacza, że istnieje jednokierunkowe połączenie z lokacji omawianego obiektu do lokacji wymienionej

jako wartość klucza `Destination`. Żeby połączenie było dwustronne, to w obu obiektach lokacji musi znajdować się odpowiedni zapis w tablicy `Connections`.

Pozostałe krawędzie grafu warstwowego, czyli krawędzie międzywarstwowe i wewnątrzwarstwowe w pozostałych warstwach łączą węzły w strukturę drzewiastą, dlatego mogą być reprezentowane poprzez zagnieżdżanie słowników. Istnienie w danym węźle klucza `Characters`, `Items` lub `Narration` oznacza, że dany węzeł jest wierzchołkiem docelowym krawędzi wychodzącej z wierzchołka będącego elementem tablicy – wartości tego klucza.

Różnice między specyfikacją grafu a instancji grafu

SHEAF używany jest do zapisu dwóch typów struktur: grafu czterowarstwowego z częściowym bądź pełnym etykietowaniem, którym jest lewa strona produkcji, oraz instancji grafu czterowarstwowego, którym jest opis stanu świata. W drugim przypadku wymagane jest zdefiniowanie klucza `Name` dla każdego węzła grafu. Druga istotna różnica dotyczy wartości słownika `ATTR-DICT`. W lewej stronie produkcji, która jest grafem a nie instancją, występowanie klucza w słowniku odnosi się do funkcji atrybutowania *attr* a nie funkcji wartości *val* i każdej nazwie atrybutu przypisywana jest wartość *null*. Zakresy konkretnych wartości definiowane są dopiero w słowniku `Preconditions`. Dla porządku należy dodać, że w samym zapisie możliwe jest zastosowanie wersji skróconej (por. rozdział 4.5). W strukturze SHEAF odpowiadającej stanowi świata możliwe jest z kolei zastosowanie wyłącznie wartości atrybutów z zakresu łańcuchów, liczb i wartości logicznych.

4.2 Referencje

Odwoływanie się do węzłów bezpośrednio dopasowanych do lewej strony

Referencja to łańcuch, który w ramach danej struktury SHEAF w jednoznaczny sposób wskazuje słownik reprezentujący wierzchołek albo krawędź grafu (czyli lokację, postać, przedmiot, element fabuły, połączenie pomiędzy lokacjami). Jako referencji można użyć:

- Etykiety wskazywanego słownika, czyli wartości związanej z jego kluczem `Name`. Aby taka forma referencji była jednoznaczna etykieta musi być unikatowa w ramach danego SHEAF.
- Identyfikatora wskazywanego słownika, czyli wartości związanej z jego kluczem `Id`. Jest to mechanizm umożliwiający rozróżnianie węzłów o identycznych etykietach i z definicji wartość klucza `Id` jest jednoznaczna w ramach struktury SHEAF.
- Łańcucha złożonego z identyfikatora albo unikatowej etykiety, znaku „/” i łańcucha `Characters`, `Items` albo `Narration`. Ta forma referencji pozwala wskazywać tablice.

Dwa pierwsze typy referencji będziemy dalej nazywać NODE-REF, a ostatnią: ARRAY-REF. Obecna wersja specyfikacji standardu nie obejmuje referencji krawędziowych. Jeżeli w przyszłości będziemy odwoływać się do krawędzi, to wprowadzimy także pojęcie CONNECTION-REF.

Odwoływanie się do węzłów ze snopków

Wiele produkcji opisujących akcje wykonywane w świecie gry będzie musiało wykonywać operacje nie tylko na słownikach, które zostały jawnie dopasowane do lewej strony produkcji, lecz również na słownikach będących ich potomkami w grafie stanu świata gry, czyli na snopkach.

Operacje na obiektach poza dopasowaniem pojawiają się np. wtedy, gdy mamy po lewej stronie jakąś postać, a operacje chcemy wykonać na wszystkich przedmiotach przez nią posiadanych. Do wskazywania takich, być może pustych, zbiorów wierzchołków zagnieżdżonych hierarchicznie wewnątrz jawnie dopasowanych wierzchołków służą multireferencje.

Multireferencja (ang. *multireference*) to łańcuch złożony z rozdzielonych znakiem „/” segmentów. Pierwszym segmentem, tak samo jak w przypadku zwykłych referencji, jest identyfikator słownika albo jego unikatowa etykieta. Następne segmenty to na przemian:

- jeden z trzech łańcuchów `Characters`, `Items` albo `Narration`, wskazujący jedną z tablic zagnieżdżonych w rozważanym słowniku,
- nazwa, wskazująca wszystkie słowniki o tej nazwie zagnieżdżone w rozważanej tablicy (może ich być zero i to nie będzie błędem).

Multireferencje jako typ danych będziemy dalej nazywać: NODE-MULTIREF.

Możliwa jest multireferencja jednosegmentowa, która sprowadza się do referencji NODE-REF.

Multireferencje służą tylko do wskazywania słowników, nie tablic. Co za tym idzie, w multireferencji skonstruowanej zgodnie z powyższymi zasadami zawsze będzie nieparzysta liczba segmentów.

Zamiast nazwy można podać znak „*”, co oznacza, że w tym miejscu może wystąpić słownik o dowolnej etykiecie (wyrażonej przez klucz `Name`). Dwuznak „**” oznacza zaś, że w tym miejscu może wystąpić dowolna liczba dowolnych segmentów (w tym zero). Pojedynczej gwiazdki można użyć w środku lub na końcu multireferencji, ale nie na jej początku.

4.3 Struktura predykatów stosowalności

Predykaty stosowalności umieszczone są w liście w kluczu **Preconditions**. Każdy predykat jest słownikiem ze ściśle określoną liczbą i zakresem kluczy. Standard definiuje dwa typy takich obiektów. Dopuszczalne jest w ramach konkretnych implementacji rozszerzanie funkcjonalności systemu o inne obiekty.

Kolejność sprawdzania predykatów nie ma znaczenia. Wartością zwracaną po sprawdzeniu każdego warunku jest *true* lub *false*. Procedura sprawdzania predykatów zwraca sumę logiczną wartości poszczególnych predykatów. Zwrócenie wartości *false* spowoduje odrzucenie produkcji mimo dopasowania lewej strony produkcji do instancji świata.

Obiekty określone w standardzie StoryGraph 1.2 to:

```
{ "Cond": "BOOLEAN-EXPRESSION" }
{ "Count": "NODE-MULTIREF", "Min": NUMBER, "Max": NUMBER }
```

- **Cond** pozwala określić zależność arytmetyczną lub logiczną pomiędzy różnymi wartościami atrybutów obiektów wskazanych w lewej stronie produkcji, gdzie parametr **BOOLEAN-EXPRESSION** jest wyrażeniem logicznym języka programowania (np. Python, GDScript),
- **Count** pozwala ograniczyć od dołu lub od góry (jedno z ograniczeń można pominąć) liczbę obiektów spełniających podaną multireferencję. Liczba słowników, które się do niej w grafie stanu świata dopasują, jest porównywana z podanymi w warunku ograniczeniami i jeśli się w nich mieści, to warunek jest spełniony.

4.4 Struktura instrukcji

Instrukcje są zebranymi w tablicę słownikami ze ściśle określoną liczbą i zakresem kluczy. Standard StoryGraph 1.2 definiuje pięć typów takich obiektów. Dopuszczalne jest w ramach konkretnych implementacji rozszerzanie funkcjonalności systemu o inne typy instrukcji.

Instrukcje zdefiniowane są poprzez klucz **Op** przyjmujący jako wartość nazwę instrukcji.

```
{ "Op": "OPERATION-NAME", ... }
```

Pozostałe parametry będą przyjmowały różne wartości, w szczególności w dalszej części wyróżnimy wartości typu **EXPRESSION**. Będziemy też definiować wartości odnoszące się do węzłów: **NODE-REF** oraz parametry związane z atrybutami: **NODE-REF.ATTR-NAME** oraz **CONNECTION-REF.ATTR-NAME**.

Typy instrukcji

Zbiór instrukcji możemy w obrębie specyfikacji dowolnie rozszerzać, opisując dodaną instrukcją nowy sposób modyfikacji dowolnego węzła lub krawędzi w GWSG. Można wszakże zauważyć, że jest kilka poziomów zależności węzłów modyfikowanych od węzłów dopasowanych do lewej strony produkcji.

Pierwszy poziom obejmuje modyfikacje ograniczone do węzłów dopasowanych do lewej strony. W takim wypadku produkcja grafowa daje się wyrazić w modelu lewa strona-prawa strona. W obecnej specyfikacji takimi operacjami są instrukcja `create` i wszystkie operacje na atrybutach oraz pozostałe operacje, jeśli parametr `Nodes` wyrażony jest multireferencją jedno-segmentową.

Drugi poziom pozwala na modyfikacje węzłów z całego snopka węzła dopasowanego do lewej strony produkcji. Obejmuje operacje `move`, `copy` oraz `delete`, jeśli parametr `Nodes` wymagany w tych operacjach wyraża się multireferencją wielosegmentową.

Trzeci poziom pozwalający na największą dowolność w kształtowaniu świata, ale wymagający wskazywania modyfikowanych węzłów bezpośrednio w instrukcji, zakłada odwołanie się do każdego węzła GWSG w sposób zdefiniowany bezpośrednio w implementacji. Operacje takie nie podlegają specyfikacji.

Poniżej opisanych jest osiem operacji z pierwszego i drugiego poziomu. Wystarczają one w zupełności do przedstawienia typowych sytuacji fabularnych, ale mają swoje ograniczenia, o których wspomniano na końcu rozdziału.

Przenoszenie słownika

Operacja `move` powoduje, że wskazany słownik, razem z zagnieżdżonymi w nim tablicami i słownikami, jest usuwany z tablicy, w której do tej pory się znajdował, i wstawiany do innej.

Operacja przenoszenia wymaga podania pary argumentów `Nodes` i `To`.

Jeśli multireferencja w parametrze `Nodes` będzie wielosegmentowa, pozwoli to przenieść wiele słowników jedną instrukcją.

Składnia:

```
{ "Op": "move", "Nodes": "NODE-MULTIREF", "To": "ARRAY-REF" }
```

Opcjonalny parametr `"Limit": NUMBER` pozwala ograniczyć przenoszenie do co najwyżej wskazanej liczby elementów. W pierwszej kolejności przenoszone będą elementy o najkrótszym rozwinięciu multireferencji w ścieżkę. Kolejności elementów o ścieżkach identycznej długości specyfikacja nie precyzuje.

Tworzenie słownika

Operacja `create` wstawia do wskazanej tablicy słownik o podanej zawartości, reprezentującej wierzchołek grafu z ewentualnymi atrybutami i zagnieżdżonymi wierzchołkami potomnymi.

Składnia:

```
{ "Op": "create", "In": "ARRAY-REF", "Sheaf": CHAR-DICT }
{ "Op": "create", "In": "ARRAY-REF", "Sheaf": ITEM-DICT }
{ "Op": "create", "In": "ARRAY-REF", "Sheaf": NARR-DICT }
```

Opcjonalny parametr `"Count": NUMBER` pozwala stworzyć w tablicy docelowej zadaną liczbę identycznych elementów.

Kopiowanie słownika

Operacja `copy` powoduje wstawienie do wskazanej tablicy kopii podanego węzła.

Składnia:

```
{ "Op": "copy", "Nodes": NODE-MULTIREF, "To": ARRAY-REF }
```

Opcjonalny parametr `"Count": NUMBER` pozwala stworzyć w tablicy docelowej zadaną liczbę kopii.

Usuwanie słownika

Operacja `delete` usuwa wskazany słownik.

Składnia:

```
{ "Op": "delete", "Nodes": "NODE-MULTIREF" }
```

Opcjonalny parametr `"Limit": NUMBER` pozwala ograniczyć usuwanie do co najwyżej wskazanej liczby elementów. W pierwszej kolejności usuwane będą elementy o najkrótszym rozwinięciu multireferencji w ścieżkę. Kolejności elementów o ścieżkach identycznej długości specyfikacja nie precyzuje.

Opcjonalne parametry `ChildrenLimiter`, `CharactersLimiter`, `ItemsLimiter` oraz `NarrationLimiter` przyjmują jedną z trzech wartości: `delete` (zachowanie domyślne), `move` lub `prohibit`. Domyślna wartość oznacza, że wraz z węzłem usuwamy wszystkie jego węzły potomne, a pozostałe wartości pozwalają węzły dzieci przenieść lub zablokować usunięcie węzła niepustego. Dodajmy, że parametr `prohibit` powoduje nieuwzględnianie danego węzła w limicie określonym przez parametr `Limit`.

Operacje na atrybutach

Są to operacje: `set`, `unset`, `mul` i `add`.

Atrybuty można tworzyć (nadając im przy tym jakąś wartość), modyfikować (czyli zmieniać wartość atrybutu) oraz usuwać. Tworzymy atrybut operacją `set`, `add` lub `mul`, modyfikujemy istniejący atrybut operacją `set` poprzez nadanie nowej wartości lub (dla atrybutów liczbowych) operacją `add`, dodając do aktualnej wartości podany składnik albo `mul`, mnożąc aktualną wartość przez podany czynnik. Usuwamy operacją `unset`. Wartości wstawiane do atrybutów mogą być stałymi podawanymi wprost w instrukcji albo mogą być obliczane na podstawie wcześniejszej wartości danego atrybutu i wartości innych atrybutów.

Składnia:

```
{ "Op": "set", "Attribute": "NODE-REF.ATTR-NAME", "Value": STRING }
{ "Op": "set", "Attribute": "NODE-REF.ATTR-NAME", "Value": NUMBER }
{ "Op": "set", "Attribute": "NODE-REF.ATTR-NAME", "Value": BOOLEAN }
{ "Op": "set", "Attribute": "NODE-REF.ATTR-NAME", "Expr": EXPRESSION }

{ "Op": "add", "Attribute": "NODE-REF.ATTR-NAME", "Value": NUMBER }
{ "Op": "add", "Attribute": "NODE-REF.ATTR-NAME", "Expr": EXPRESSION }

{ "Op": "mul", "Attribute": "NODE-REF.ATTR-NAME", "Value": NUMBER }
{ "Op": "mul", "Attribute": "NODE-REF.ATTR-NAME", "Expr": EXPRESSION }

{ "Op": "unset", "Attribute": "NODE-REF.ATTR-NAME" }
```

*

Mając zdefiniowaną strukturę grafu wielosnopkowego, strukturę predykatów stosowalności i strukturę instrukcji możemy już zapisać każdą produkcję, czyli trójkę (M, C, P) , gdzie M jest grafem czterowarstwowym wielosnopkowym z częściowym bądź pełnym etykietowaniem, C jest procedurą sprawdzania predykatów stosowalności opartą o tablicę predykatów a P jest procedurą modyfikującą przekształcany graf opartą o tablicę instrukcji.

Struktura opisująca graf hierarchii produkcji nie jest narzucona w specyfikacji, zdefiniowana są wyłącznie jej funkcje: ustalanie częściowego porządku i wykluczania produkcji przysłaniających ze zbioru produkcji. Wewnętrzna struktura grafu hierarchii produkcji jest pozostawiona w gestii twórców implementacji.

4.5 Format zapisu danych

Wszystkie powyższe struktury zapisane są jako obiekty JSON. Obiekt zgodnie ze specyfikacją formatu⁷ składa się z par: *nazwa: wartość*. Nazwy muszą być unikatowym łańcuchem znaków. Wartość może być dowolnym typem JSON: łańcuchem tekstowym (ang. *string*), liczbą (ang. *number*), wartością logiczną (ang. *boolean*), tablicą (ang. *array*), obiektem (ang. *object*).

W zapisie JSON produkcja jest obiektem składającym się z następujących elementów przedstawionych w przykładzie kodu 1:

Kod 1. Struktura produkcji w zapisie JSON.

```
{
  "Title": "Nazwa produkcji",
  "TitleGeneric": "Nazwa produkcji bazowej dla danej produkcji",
  "Description": "Opis produkcji wyjaśniający ją i wykorzystywany przy tworzeniu gier
  typu tekstowego.",
  "Override": liczba z zakresu 0-2,
  "LSide": {},
  "Preconditions": [],
  "Instructions": []
}
```

Podstawą opisu są: słownik o kluczu `Locations`, odwzorowujący strukturę SHEAF, czyli reprezentacja grafu wielosnopkowego, tablica o kluczu `Preconditions` odpowiadająca predykatom stosowalności i tablica o kluczu `Instructions` opisująca procedurę modyfikacji. Opisane są one w rozdziałach 4.1, 4.3 i 4.4.

Rola kluczy `Title`, `TitleGeneric`, `Description` oraz `Override` nie wynika bezpośrednio z definicji grafu *n*-warstwowego, ale jest związana z rozróżnianiem produkcji, ustalaniem częściowego porządku i wprowadzeniem mechanizmu przysyłania produkcji opisanym w rozdziale 6.1.

Przykładowa produkcja zawierająca wszystkie powyżej opisane elementy przedstawiona jest w przykładzie kodu 2 i opisuje proces sprzedaży jakiegoś przedmiotu jednemu bohaterowi przez drugiego.

⁷ RFC8259

Kod 2. Przykładowa produkcja.

```

{
  "Title": "Making a deal / Transakcja kupna-sprzedaży",
  "TitleGeneric": "Item acquisition from another character / Przejęcie przedmiotu od
  innej postaci",
  "Description": "«Seller» sprzedaje kupującemu («Buyer») przedmiot («Something»).",
  "Override": 0,
  "LSide": {
    "Locations": [
      {
        "Id": "Somewhere",
        "Attributes": {},
        "Characters": [
          {
            "Id": "Buyer",
            "IsObject": true,
            "Attributes": {
              "Money": null
            }
          },
          {
            "Id": "Seller",
            "IsObject": true,
            "Attributes": {
              "Money": null
            }
          },
          {
            "Items": [
              {
                "Id": "Something",
                "Attributes": {
                  "Value": null
                }
              }
            ]
          }
        ]
      }
    ]
  }
},

```

```

"Preconditions": [
  {
    "Cond": "Buyer.Money >= Something.Value"
  }
],
"Instructions": [
  {
    "Op": "move",
    "Nodes": "Something",
    "To": "Buyer/Items"
  },
  {
    "Op": "set",
    "Attribute": "Buyer.Money",
    "Expr": "Buyer.Money - Something.Value"
  },
  {
    "Op": "set",
    "Attribute": "Seller.Money",
    "Expr": "Seller.Money + Something.Value"
  }
]
}

```

Sposób wykorzystania parametru **Count** przedstawiony jest w przykładzie kodu 3. Przedstawione są tam trzy przypadki zliczania obiektów: liczby bohaterów w danej lokacji (oczekujemy co najwyżej dwóch postaci), liczby mieczy dostępnych bezpośrednio w ekwipunku bohatera (mamy nadzieję na co najmniej dwa) oraz liczby mieczy dostępnych dla bohatera zarówno bezpośrednio w ekwipunku jak i zagnieżdżonych w dowolnym z jego węzłów potomnych (akceptujemy nie mniej niż dwa, ale nie więcej niż cztery miecze). Wykorzystujemy w tych predykatach multireferencje omówione w rozdziale 4.2.

Kod 3. Przykład wykorzystania predykatu stosowalności do narzucenia ograniczenia na liczbę obiektów.

```

{ "Count": "Somewhere/Characters/*", "Max": 2 }
{ "Count": "Hero/Items/Sword", "Min": 2 }
{ "Count": "Hero/**/Items/Sword", "Min": 2, "Max": 4 }

```

Skrócony zapis prostych predykatów

Wyjaśnienia wymaga skrócony zapis niektórych predykatów umożliwiający dołączenie niektórych warunków bezpośrednio do zapisu lewej strony produkcji. Zgodnie z definicją lewa strona produkcji jest grafem n -warstwowym, czyli definiując ją wymagamy dopasowania na poziomie funkcji atrybutowania (funkcja *attr*, por. rozdział 3.1, *Definicja 2*) a nie wartości przypisanych tej funkcji w świecie (czyli instancji). Konkretnie zakresy wartości, czyli podzbiory przeciwdziedziny funkcji *val* (por. rozdział 3.1, *Definicja 3*) definiujemy w tablicy predykatów. W przykładach kodu 4 i 5 pokazane są sposoby zdefiniowania atrybutów `IsHigh` oraz `Height` i określenia dopuszczalnych wartości odpowiednio jako zbioru jednoelementowego lub zakresu ograniczonego od dołu.

Kod 4. Przykład wykorzystania atrybutu (wartość funkcji *attr* zadana w lewej stronie produkcji) z wymaganą konkretną wartością (zdefiniowaną w predykatach).

```
"LSide": {
  "Locations": [ {
    "Id": "Somewhere",
    "Characters": [ {
      "Name": "Hero",
      "Attributes": {
        "IsHigh": null
      }
    } ]
  } ]
},
"Preconditions": [
  { "Cond": "Hero.IsHigh == true" }
]
```

Kod 5. Przykład wykorzystania atrybutu (wartość funkcji *attr* zadana w lewej stronie produkcji) z wymaganą wartością z pewnego zakresu (zdefiniowanego w predykatach).

```
"LSide": {
  "Locations": [ {
    "Id": "Somewhere",
```

```

    "Characters": [ {
      "Name": "Hero",
      "Attributes": {
        "Height": null
      }
    } ]
  } ]
},
"Preconditions":[
  { "Cond": "Hero.Height >= 180" }
]

```

Jak widzimy na przykładach, z przeciwdziedziny funkcji *val* w pierwszym przypadku wybieramy konkretną wartość a w drugim zakres wielu wartości. Tylko w tym pierwszym przypadku możemy zastosować zapis skrócony przedstawiony w przykładzie kodu 6.

Kod 6. Przykład wykorzystania atrybutu (wartość funkcji *attr* zadana w lewej stronie produkcji) z wymaganą konkretną wartością (zdefiniowaną w zapisie skróconym w obrębie lewej strony produkcji).

```

"LSide": {
  "Locations": [ {
    "Id": "Somewhere",
    "Characters": [ {
      "Name": "Hero",
      "Attributes": {
        "IsHigh": true
      }
    } ]
  } ]
},
"Preconditions":[]

```

4.6 Ograniczenia przyjętych rozwiązań

Zarówno założenia poczynione na poziomie modelu teoretycznego jak i założenia dodawane podczas tworzenia specyfikacji systemu wpływają na możliwości tworzenia rozwiązań fabularnych. Dodatkowo możemy wprowadzić pewne ograniczenia na etapie kodowania rozwiązań implementacyjnych, istotne jednak, żeby te ograniczenia nie wpływały na możliwości kształtowania fabuły dopuszczalne w specyfikacji.

Ograniczenia wynikające z modelu formalnego i specyfikacji

Niemożliść łączenia węzłów niebędących lokacjami w cykle

Zwróćmy uwagę, że z definicji modelu formalnego wynika, że nie możemy dopuścić do umieszczenia słownika z kluczem `Connections` w węźle innym niż z warstwy lokacji, czyli warstwy wyróżnionej zgodnie z *Definicją 8* (por. rozdział 3.1). Jest to związane z warunkiem: „z każdego wierzchołka nienależącego do zbioru wierzchołków grafu $G_{ch(x_R)}$ wychodzi dokładnie jedna krawędź wewnętrzna lub zewnętrzna”. Ograniczenie to jest związane z modelem struktury grafowej jako lasu opartego na bazie dowolnie połączonych węzłów korzeni snopków, ale tworzy także dodatkową korzyść na gruncie implementacyjnym. Dodanie takiego klucza dla węzłów innych niż lokacje groziłoby łatwym rozpójnieniem grafu świata związanym z możliwością usuwania węzłów, która nie dotyczy węzłów lokacji. Dzięki ograniczeniu narzuconemu w modelu formalnym nie musimy tworzyć procedury uspójniania świata po każdym usunięciu węzła.

Niemożliść przenoszenia węzłów między warstwami

Definicja modelu wymusza także rozłączność zbioru etykiet węzłów z różnych warstw dla danego stanu GWSG, co jest związane z warunkiem w *Definicji 4*: „zbiory użytych etykiet, czyli $lab_{ch(x)}(V_{ch(x)})$, są rozłączne”. W konsekwencji oznacza to rozłączność zbiorów etykiet na poziomie definiowania zbiorów etykiet.

Niemożliść dynamicznego definiowania atrybutów nowo tworzonych węzłów

Z kolei na poziomie specyfikacji wyłącznie ograniczenia zapisu parametrów instrukcji `create` powodują, że nie da się wartości atrybutów nowo tworzonych węzłów i węzłów ze snopka uzależnić od wartości węzłów dopasowanych w lewej stronie produkcji. Nie da się też do nowo utworzonych węzłów odwoływać przez referencje tablicowe. W kolejnych wersjach specyfikacja być może zostanie poszerzona o możliwość wyliczania wartości atrybutów węzłów nowo tworzonych analogicznie jak w przypadku zmiany wartości atrybutów dla wcześniej istniejących węzłów.

Jakie konsekwencje niosą za sobą powyższe ograniczenia? Przyjrzyjmy się produkcji, na której kształt istotny wpływ mają powyższe ograniczenia. Wyobraźmy sobie przekształcenie zabitego wilka w zwłoki:

Kod 7. Produkcja przekształcająca postać o etykiecie „Wolf” i atrybucie „IsDead” z wartością *true* w przedmiot.

```
{
  "Title": "Turning a dead wolf into a pelt / Zamiana martwego wilka w skórę",
  "TitleGeneric": "Turning a dead character into an item / Zamiana martwej postaci
  w przedmiot",
  "Description": "Martwy wilk znika a na jego miejscu pojawia się jego skóra.",
  "Override": 1,
  "LSide": {
    "Locations": [
      {
        "Id": "Somewhere",
        "Characters": [
          {
            "Name": "Wolf",
            "Attributes": {
              "IsDead": true
            }
          }
        ]
      }
    ],
  },
  "Instructions": [
    {
      "Op": "create",
      "In": "Somewhere/Items",
      "Sheaf": {
        "Name": "Wolf_pelt",
      }
    },
    {
      "Op": "delete",
      "Nodes": "Wolf"
    }
  ]
}
```

Nie możemy przenieść martwego wilka do przedmiotów po nadaniu mu atrybutów przedmiotu (`Value`) i usunięciu atrybutów postaci (`HP`, `Money`), ponieważ model zakłada rozłączność alfabetów etykiet. Usuwamy w takim razie bohatera i tworzymy przedmiot. Nie możemy niestety uzależnić wartości atrybutu `Value` od wcześniejszego atrybutu `HP` ze względu na sposób zapisu struktury CHAR-DICT, ITEM-DICT i NARR-DICT. Także jeśli martwy wilk miał jakiegokolwiek przedmioty lub postaci, np. był to klasyczny wilk z bajki *Czerwony Kapturek*, który zjadł babcię i wnuczkę, i koszyk, i kanapki, to po wykonaniu takiej produkcji znikają one na zawsze. Nie możemy ich przenieść do świeżo utworzonego obiektu `Wolf_pelt`, ponieważ jako niedopasowany do lewej strony nie może być, zgodnie ze specyfikacją, obiektem składowym ARRAY-REF.

Możemy oczywiście poradzić sobie z tą niedogodnością. W grze przykładowej autorzy proponowali rozwiązanie fabularne: wszystkie przedmioty i postaci związane z bohaterem wypadają do lokacji już w momencie śmierci, więc martwy bohater nie ma żadnych przedmiotów, które zostałyby usunięte ze świata, gdy zamieni się w zwłoki.

Konieczność wskazania podmiotu akcji

Istotnym elementem ograniczającym dowolność w stosowaniu produkcji jest wymóg, aby wszystkie produkcje z wyjątkiem produkcji automatycznych miały wskazanego za pomocą parametru `IsObject` przynajmniej jednego bohatera jako podmiot produkcji. Dozwolone jest, aby więcej niż jedna postać mogła inicjować produkcję, lub inaczej – aby bohater aktywny (sterowany przez gracza) mógł wykonać daną produkcję z pozycji więcej niż jednej roli. Np. produkcja transakcji kupna-sprzedaży może być inicjowana przez gracza zarówno z pozycji kupującego jak i sprzedającego, ale wskazanie podmiotu produkcji nie może być pominięte. W implementacji można ominąć to ograniczenie, np. w implementacji referencyjnej opisanej w rozdziale 5.3 można wywołać tryb ignorowania tego klucza i pozwalać użytkownikowi wybierać dowolną produkcję albo wręcz ignorować wskazanie bohatera i wyliczać dostępne produkcje dla wszystkich postaci na mapie świata, ale jest to działanie kosztowne obliczeniowo i dokonywane na odpowiedzialność twórców implementacji.

Bezpotomność węzłów narracji

Kolejnym ograniczeniem wynikającym ze specyfikacji jest ustanowienie węzłów narracyjnych jako liści snopka. Nie mogą one zawierać już żadnych zagnieżdżonych węzłów.

Konieczność zdefiniowania tekstowego identyfikatora dla elementów nierozróżnialnych za pomocą etykiety

W modelu grafowym traktujemy każdy wierzchołek grafu jako obiekt z definicji niepowtarzalny i odróżnialny od pozostałych wierzchołków. W trakcie działania programu opartego na modelu grafowym osiągamy ten sam efekt, ponieważ każdy obiekt jest rozróżnialny przez adres w pamięci, pod którym się znajduje, i wystarczy odwoływać się do obiektów przez wskaźniki, alby uzyskać modelową rozróżnialność. W specyfikacji zapisu stanu świata do formatu JSON musimy jednak zadbać ręcznie o rozróżnialność węzłów grafu. Będzie to wykorzystywane w dwóch sytuacjach: po pierwsze podczas zapisywania aktualnego stanu świata do pliku w trakcie rozgrywki, po drugie do wskazania, który z węzłów dopasowanych do lewej strony produkcji modyfikujemy w danej instrukcji. W jednym i drugim przypadku skorzystamy z mechanizmu identyfikatora (dodatkowego klucza **Id**). W przypadku zapisu stanu rozgrywki do pliku identyfikatory tworzone są automatycznie, np. na podstawie adresu w pamięci, w przypadku tworzenia produkcji identyfikator jest łańcuchem tekstowym wybranym przez projektanta, zazwyczaj powiązany z funkcją danego węzła w akcji (np. „Buyer”, „Seller”, „Authority”). Rozróżnialność identyfikatorów ma znaczenie wyłącznie w obrębie jednego grafu. Dodatkowo, w przypadku produkcji definiowanie **Id** nie jest wymagane, jeśli węzeł jest rozróżnialny za pomocą etykiety, tzn. jest jedynym węzłem o danej etykiecie w lewej stronie produkcji albo jeśli nie występuje w żadnym predykanie ani w instrukcji.

Konieczność zdefiniowania nazwy produkcji i podania jej opisu słownego

Identyfikowanie produkcji za pomocą nazwy (klucz **Title**) wprowadzono z tego samego powodu co identyfikatory węzłów – w celu umożliwienia odwołania się do konkretnej produkcji. Dodatkowo użytkownik systemu na podstawie nazwy i opisu (klucz **Description**) może dowiedzieć się, jakiej akcji dana produkcja dotyczy.

Ograniczenia możliwe do określenia na poziomie implementacji

Na poziomie implementacji możliwe jest zarówno rozszerzanie możliwości systemu, np. dawanie instrukcji nie wyspecyfikowanych jako podstawowe ale też narzucanie dodatkowych ograniczeń. Np. w implementacji referencyjnej opisanej w rozdziale 5.3 narzucony został warunek, aby lokacja zawierająca postać z kluczem **IsObject** znajdowała się w pierwszej lokacji w tablicy lokacji. Nie ogranicza to możliwości projektanta, bo kolejność lokacji nie ma znaczenia fabularnego, a ułatwia proces dopasowania generycznego.

Dodatkowo też na poziomie implementacji możliwe są pewne ograniczenia porządkowe zwiększające czytelność zapisu produkcji czy umożliwiające komunikowanie się z użytkownikiem, np. wymóg umieszczania **Id** węzłów pozbawionych etykiety w opisie słownym produkcji w formacie umożliwiającym ich automatyczną podmianę po wybraniu wariantu dopasowania i wyświetlenie użytkownikowi takiego zindywidualizowanego komunikatu.

5 Implementacja

Specyfikacja systemu StoryGraph umożliwia definiowanie świata gry i akcji modyfikujących graf. Może być wykorzystana w systemach wspomagających projektowanie gry albo do zbudowania silnika fabularnego gry przygodowej. Prowadzenie rozgrywki będzie wymagało naprzemiennego przedstawiania graczowi aktualnego stanu świata i wykonywania wskazanych przez niego działań. Akcje gracza mogą być przeplatane działaniami postaci niezależnych i akcjami automatycznymi, czyli wydarzeniami wynikającymi z zasad funkcjonowania świata nie inicjowanymi przez żadną postać, wynikającymi bezpośrednio i nieuchronnie z zaistnienia odpowiednich okoliczności.

Każda akcja reprezentowana jest przez produkcję grafową, czyli trójkę (M, C, P) , gdzie M jest grafem czterowarstwowym z częściowym bądź pełnym etykietowaniem, C jest procedurą sprawdzania predykatów stosowalności, a P jest procedurą modyfikującą przekształcany graf.

Świat reprezentowany jest przez instancję wielosnopkowego grafu czterowarstwowego.

*

Implementacja referencyjna przywoływana w pracy znajduje się w repozytorium:
<https://github.com/iwonagg/StoryGraphPhD>.

5.1 Algorytmy stworzone na potrzeby implementacji

Znajdowanie dopasowania generycznego grafu czterowarstwowego

Najciekawszym algorytmicznie i najkosztowniejszym obliczeniowo elementem implementacji modelu grafowego jest znajdowanie dopasowania generycznego grafu czterowarstwowego. W ogólnym przypadku problem znajdowania podgrafu izomorficznego z wzorcowym jest NP-zupełnym problemem decyzyjnym, ale ograniczenie struktury grafowej do grafu wielosnopkowego ze wskazanym węzłem podmiotu produkcji pozwoliło na zaproponowanie algorytmu dopasowanego do specyfiki danych fabularnych.

Ułatwienia wynikające ze specyfiki struktur fabularnych:

- Podmiotem produkcji jest gracz, który znajduje się w konkretnej lokacji, więc dwa węzły (lokacja i postać) mamy dopasowane za darmo.
- Lewe strony produkcji są zazwyczaj niewielkimi grafami, ponieważ odwzorowują działania bohatera, które zazwyczaj obejmują interakcje z jedną-dwoma innymi postaciami, jednym-kilkoma przedmiotami i krótkie ścieżki.
- Podejmując akcję opieramy się na tym, co widzimy wokół, dlatego zazwyczaj tylko snopek głównej lokacji jest rozbudowany.

Wyzwaniem przy tworzeniu algorytmu jest z kolei to, że lewa strona produkcji może być częściowo etykietowana, co powoduje, że w każdej warstwie musimy rozpatrywać podzbiór węzłów potencjalnie pasujących do wszystkich etykiet danej warstwy w świecie, a także że etykiety węzłów nie są unikalne i że nie poprzestajemy na pierwszym pasującym podgrafie, tylko wyszukujemy wszystkie.

Danymi wejściowymi algorytmu są dwa grafy: podgraf, którego dopasowania szukamy i graf, w którym chcemy znaleźć dopasowanie, a wynikiem lista list par wierzchołków: pierwszy z pierwszego a drugi z drugiego grafu.

Problem dopasowania grafu n -warstwowego rozbijamy na dwie części bazując na istnieniu warstwy wyróżnionej w grafie wielosnopkowym. W pierwszym etapie (1. a-e) szukamy dopasowania warstwy lokacji a w drugim (2.) rekurencyjnie sprawdzamy dopasowanie dzieci.

Procedura ta odpowiada definicji dopasowania generycznego grafu / instancji grafu n -warstwowego z częściowym bądź pełnym etykietowaniem. W pierwszym etapie szukamy dopasowania generycznego z rodziny dopasowań generycznych $\{k_x\}_{x \in X_M}$ dla x z etykietą "Locations", czyli szukamy zbioru par wierzchołków zachowujących własności wierzchołków i krawędzi (por. rozdział 3.1, *Definicja 16*):

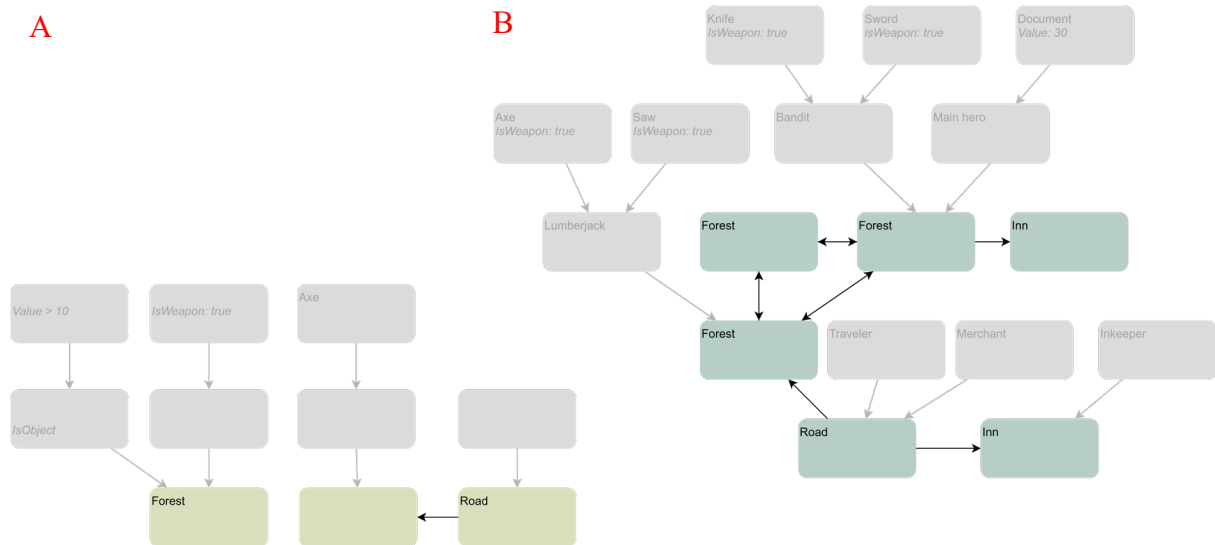
- $\forall e \in E_K : k_V(s_K(e)) = s_Q(k_E(e)),$
- $\forall e \in E_K : k_V(t_K(e)) = t_Q(k_E(e)),$
- $\forall v \in V_K : v \in \text{dom}(\text{lab}_K) \Rightarrow \text{lab}_K(v) = \text{lab}_Q(k_V(v)),$
- $\forall v \in V_K : \text{attr}_K(v) \subseteq \text{attr}_Q(k_V(v)).$

Ostatni warunek dopasowania z *Definicji 16* (warunek dopasowania atrybutów krawędzi) pomijamy, bo w tej implementacji krawędzie nie mają atrybutów.

W kolejnych krokach algorytmu usuwamy pary wierzchołków niespełniające kolejnych warunków aż pozostaną tylko pary stanowiące poprawne dopasowanie zgodnie z definicją.

Poniżej przedstawiono schemat działania algorytmu dla przykładowych danych i zwizualizowano poszczególne etapy (rysunki 13–19).

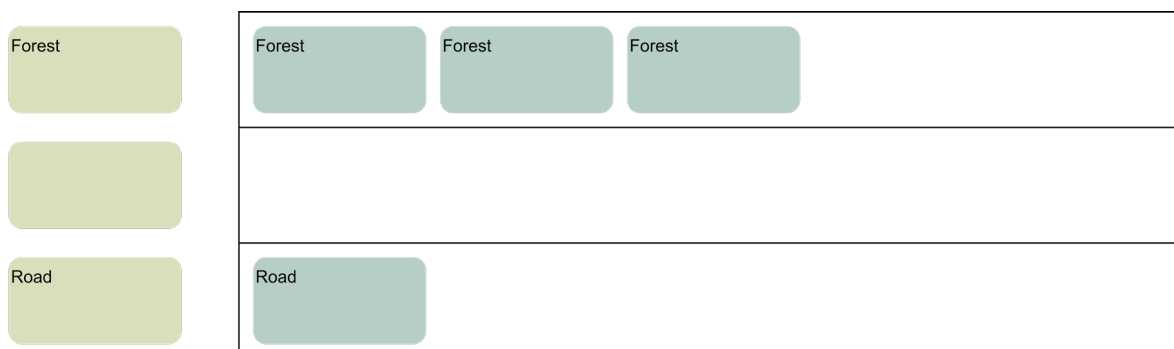
1. Szukamy dopasowania podgrafu lewej strony w warstwie lokacji grafu świata:



Rys. 13. Przykładowe dane wejściowe algorytmu dopasowania. Na zielono interesujące nas w pierwszym etapie algorytmu węzły warstwy lokacji. Na szaro ignorowane pozostałe węzły. a) Lewa strona produkcji: warstwa lokacji składa się z trzech węzłów (kolor jasnozielony), b) Świat: warstwa lokacji składa się z sześciu węzłów (kolor ciemnozielony).

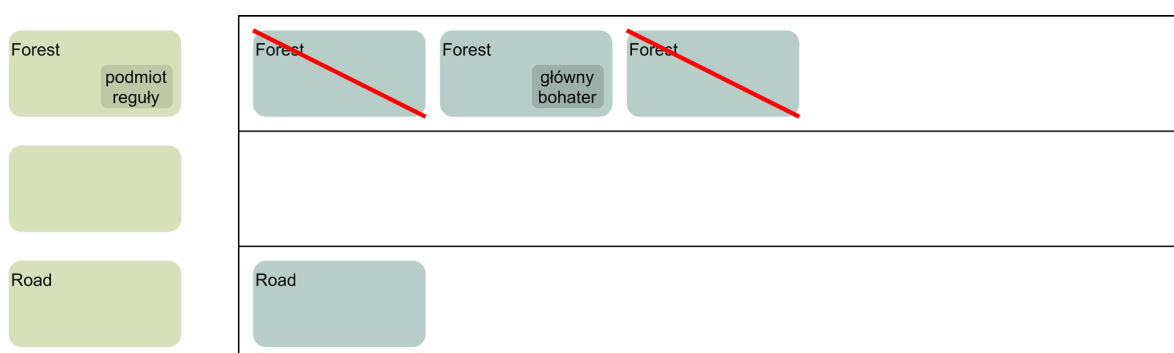
a) Tworzymy wstępną listę dopasowań węzłów na podstawie częściowego etykietowania:

$$\forall v \in V_K : v \in \text{dom}(\text{lab}_K) \Rightarrow \text{lab}_K(v) = \text{lab}_Q(kv(v)).$$



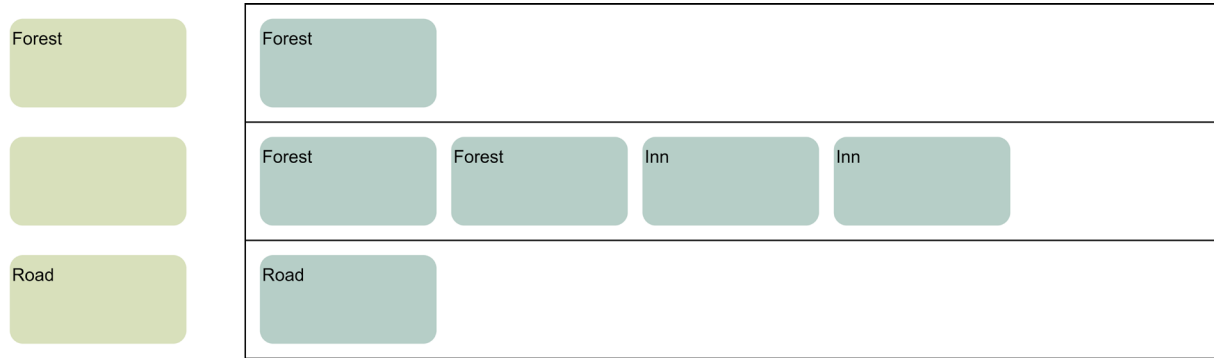
Rys. 14. Pierwszy krok dopasowania warstwy lokacji: przypisanie lokacjom lewej strony posiadającym etykiety listy lokacji węzłów świata ograniczonych przez daną etykietę.

b) Ujednoznaczniamy pozycję gracza. Jeśli postać gracza przypisana jest w lewej stronie produkcji lokacji nienależącej do $\text{dom}(\text{lab}_K)$ rozszerzamy funkcję przypisując temu węzłowi etykietę odpowiadającą etykietcie lokacji gracza w świecie. Pozostałe lokacje o tej nazwie traktowanie są analogicznie do wszystkich innych etykietowanych węzłów i rozważane są jako potencjalne dopasowania węzłów o takiej samej etykietcie w lewej stronie bądź, jeśli nie spełniają warunku równoliczności z węzłami o tej nazwie w świecie (z pominięciem lokacji, w której jest gracz), zasilają zbiór węzłów niewykorzystanych.



Rys. 15. Drugi krok dopasowania warstwy lokacji: lokacji zawierającej postać wskazaną jako podmiot reguły jednoznacznie przypisujemy lokację w świecie zawierającą postać gracza. Pozostałe lokacje o tej nazwie znajdują się w puli lokacji niewykorzystanych.

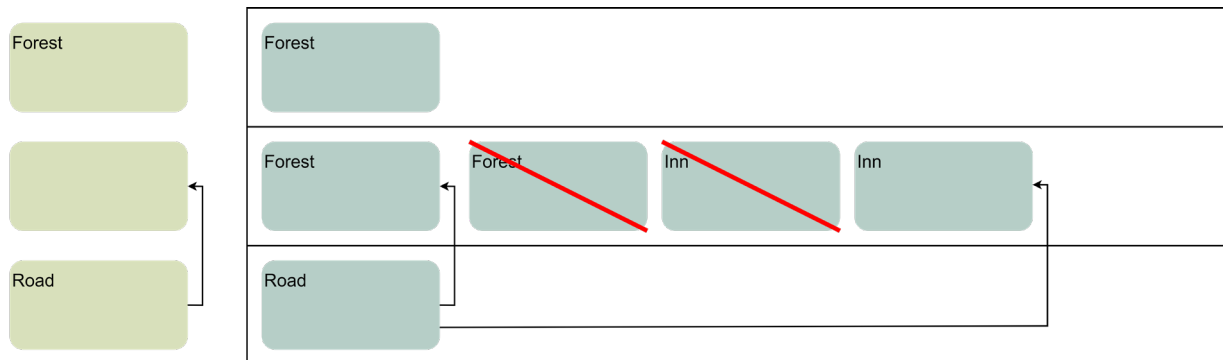
- c) Przypisujemy węzłom $v \notin \text{dom}(\text{lab}_K)$ węzły niewykorzystane w poprzednim punkcie. Niewykorzystane tzn. takie, że $\forall l \in L$ liczba węzłów z etykietą l jest większa w świecie niż w lewej stronie produkcji.



Rys. 16. Trzeci krok dopasowania warstwy lokacji: lokacjom lewej strony nie identyfikowanym przez etykiety przypisujemy listę lokacji węzłów świata niewykorzystanych w poprzednim kroku. Zwróćmy uwagę, że z trzech lokacji o etykiecie „Forest” przypisaliliśmy tylko dwie, bo jedna jest jednoznacznie wykorzystana w linijce wyżej. Nie przypisaliliśmy też lokacji „Road”, ponieważ liczność węzłów o takiej etykiecie jest równa w lewej stronie i w świecie.

- d) Usuwamy z listy dopasowań węzły niespełniające warunku zachowania sąsiedztwa:

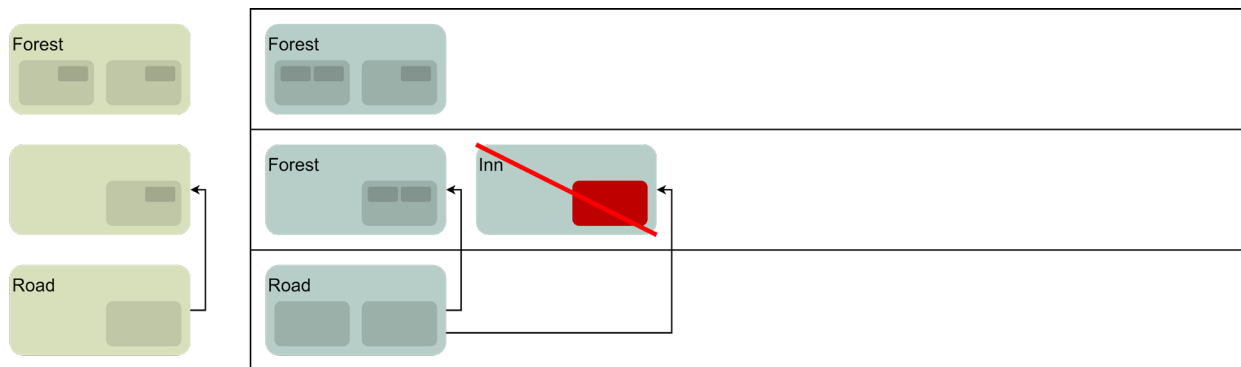
$$\forall e \in E_K : k_V(s_K(e)) = s_Q(k_E(e)) \text{ oraz } \forall e \in E_K : k_V(t_K(e)) = t_Q(k_E(e))$$



Rys. 17. Czwarty krok dopasowania warstwy lokacji: usuwamy z listy potencjalnych dopasowań lokacje niespełniające warunku sąsiedztwa.

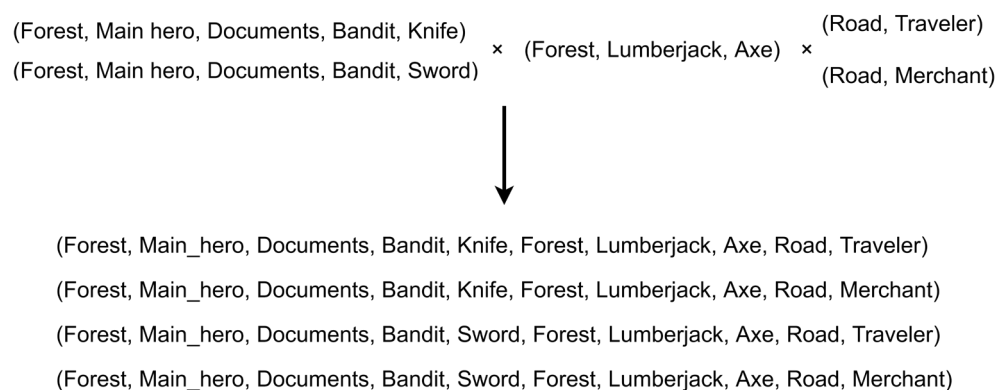
- e) Usuwamy z listy dopasowań węzły niespełniające warunku zachowania atrybutów: $\forall v \in V_K : \text{attr}_K(v) \subseteq \text{attr}_Q(k_V(v))$. Dla ścisłości należy dodać, że w tym miejscu sprawdzamy także częściowo warunki stosowalności, te, w których warunek bazuje na wartościach pojedynczego węzła. Spowodowane jest to kwestiami optymalizacyjnymi.

2. Korzystamy z własności grafu wielosnopkowego, jakim jest możliwość wydzielenia podgrafów snopkowych i ścieżek w tych podgrafach i rekurencyjnie sprawdzamy zgodność warunków dopasowania generycznego. Niezgodność któregośkolwiek elementu ścieżki pozwala usunąć całą ścieżkę a w konsekwencji cały snopek z listy alternatyw.



Rys. 18. Rekurencyjne zagłębienie algorytmu. Dla każdego snopka wykonujemy dopasowanie w sposób analogiczny do przedstawionego powyżej z dokładnością do procedury sprawdzania sąsiadów, których nie ma.

Dzięki temu, że znalezienie dopasowania gwarantuje nam odpowiedniość struktur grafowych, to jako wynik zwrócony przez funkcję dopasowania możemy zwracać listę możliwych dopasowań, czyli listę list par wskaźników na węzły z dopasowanego grafu (lewa strona produkcji) i instancji (świat).



Rys. 19. Iloczyn kartezjański wyników funkcji wywołanej rekurencyjnie.

Poniżej przedstawione są schematycznie etapy działania algorytmu (przykład kodu 8).

Kod 8. Pseudokod przedstawiający etapy działania algorytmu.

```
func is_matching_world( production ):
    if player_character_at_location ( production.locations ):

        matching_locations = world_locations

        for location in matching_locations:
            if check_for_explicit_location_match( production.locations, location ) == false:
                if check_for_multilocation_match( production.locations, location ) == false:
                    matching_locations.remove( location )

        match_pairs = []
        for location in matching_locations:
            has_match = false
            for location_p in production.locations:
                if is_recursive_matching( location_p.children, location.children ) == true:
                    match_pairs.append( location_p, location )
                    has_match = true

            if has_match == false:
                matching_locations.remove( location )

        for location_p in production.locations:
            if match_pairs.get( location_p ).size() == 0:
                return false
        return match_pairs

    return false
```

Budowa drzewa hierarchii produkcji

Na bazie zdefiniowanego zbioru produkcji budowana jest częściowo uporządkowana struktura grafu drzewa hierarchii produkcji.

Algorytm budowania drzewa hierarchii opiera się na trzech podstawach: 1) wskazaniu przez projektanta produkcji bazowej dla danej produkcji, 2) wartości parametru **Override** oraz 3) zjawisku podobieństwa produkcji.

Przez podobieństwo produkcji rozumiemy sytuację, kiedy dopasowanie jednej produkcji zawsze oznacza dopasowanie drugiej. Nie jest to relacja zwrotna, ale jest przechodnia.

Dla danej produkcji możemy znaleźć więcej niż jedną podobną produkcję, graf takich zależności będzie skierowanym grafem acyklicznym. Budujemy podgraf częściowy tego grafu pozostawiając tylko krawędzie wyznaczone parami `Title` – `TitleGeneric` i uzyskujemy drzewo hierarchii.

Produkcja bazowa dla danej produkcji określana jest przez projektanta w kluczu `TitleGeneric`. Nie może to być działanie zautomatyzowane, ponieważ wskazanie produkcji musi być świadomą decyzją projektową mającą konsekwencje przy wykonywaniu działań w trakcie rozgrywki, w szczególności niekiedy akcje bardziej szczegółowe obejmują działania, do których jest podobne kilka produkcji generycznych i projektant musi wybrać, która z nich jest bazowa dla tworzonej właśnie produkcji szczegółowej. Będzie to istotne dla określania procedury przysłaniania produkcji, która pozwoli zachować spójność fabularną w sytuacji dostępności kilku produkcji o różnych poziomach szczegółowości, co jest opisane w rozdziale 6.1, podrozdział *Przesłanianie produkcji*.

Projektant może wybrać produkcję bazową tylko spośród zbioru produkcji podobnych, dlatego więc niezbędna jest procedura sprawdzania podobieństwa produkcji. Docelowo w systemie wspomagającym tworzenie produkcji będzie możliwość wyboru produkcji bazowej w kreatorze produkcji i uzupełnienie jej o elementy dodatkowe bez możliwości modyfikacji bazowych wierzchołków i krawędzi innej niż dodanie etykiet i atrybutów, co zapobiegnie powstawaniu błędów. W momencie składania niniejszej pracy ta funkcja systemu jest jeszcze w fazie testów, więc dostępna jest wyłącznie usługa walidacji poprawności wskazania produkcji bazowej (zawierania się jej w zbiorze produkcji podobnych) i wizualizacji drzewa produkcji. Usługa walidacji będzie potrzebna nawet w momencie udostępnienia generatora produkcji, ponieważ nie możemy wykluczyć ręcznej modyfikacji plików źródłowych poza systemem.

Sprawdzanie podobieństwa dwóch produkcji

Aby można było mówić o podobieństwie produkcji, lewa strona produkcji pochodnej musi zawierać wierzchołki i krawędzie dające się dopasować do elementów produkcji bazowej. Mogą być bardziej uszczegółowione, np. węzeł produkcji szczegółowej może posiadać etykietę lub konkretny atrybut. Może też być poszerzona o dodatkowe węzły. Wymóg ten nie dotyczy instrukcji, choć najczęściej zbiór instrukcji produkcji pochodnych zawiera w sobie instrukcje produkcji bazowej i ewentualnie inne instrukcje manipulujące elementami poszerzającymi lewą

stronę produkcji bazowej. Zdarzają się jednak sytuacje, w których trzymanie się ściśle tej zasady nie ma uzasadnienia fabularnego. Do celów późniejszego tworzenia listy produkcji dostępnych dla gracza wymagana jest bezwzględnie tylko odpowiedniość lewych stron.

Od strony algorytmicznej w procesie sprawdzania podobieństwa produkcji bazujemy na opisanym w poprzednim rozdziale algorytmie dopasowywania produkcji do świata, tym razem jednak nie szukamy dopasowania do GWSG, ale do lewej strony produkcji pochodnej. Jeśli znajdziemy choć jedno dopasowanie – produkcja jest podobna. Jeśli znaleźliśmy kilka możliwych dopasowań, to możemy ograniczyć wybór sprawdzając, dla którego wariantu dopasowania podobny jest także zbiór instrukcji. Brak podobieństwa instrukcji nie wyklucza jednak podobieństwa produkcji jako całości.

Przykład podobieństwa produkcji pokazany jest na rysunku 20 i 21.



Rys. 20. Podobieństwo lewej strony produkcji bazowej i szczegółowej dziedziczącej po niej.



Rys. 21. Podobieństwo zbioru instrukcji produkcji bazowej i szczegółowej dziedziczącej po niej.

5.2 Dane wejściowe

Dane wejściowe w formacie JSON zawierają informacje o strukturze i własnościach świata początkowego oraz listę produkcji odpowiadającą akcjom dostępnym w tym świecie. Przy inicjalizacji wskazywany jest bohater kierowany przez gracza, czyli słownik odpowiadający jednemu wierzchołkowi grafu warstwowego z warstwy „Characters”.

5.3 Architektura implementacji referencyjnej

Implementacja referencyjna systemu StoryGraph została stworzona przy użyciu języka Python w wersji 3.9. Przygotowanie projektu w takiej architekturze zapewnia jego wieloplatformowość i możliwość uruchomienia na najpopularniejszych systemach operacyjnych, jak Windows, Linux czy MacOS.

Kod źródłowy projektu jest przechowywany w repozytorium Git hostowanym na platformie GitHub.

Ze względu na wiele komponentów projektu, takich jak:

- końcówki REST API (ang. REST API *endpoints*) zapewniające walidację i wizualizację danych dla użytkowników nieposiadających dostępu do kodu źródłowego projektu,
- narzędzia do walidacji danych oraz struktury plików produkcji w kontekście schematu JSON Schema zdefiniowanego w specyfikacji StoryGraph 1.2. (por. Dodatek C),
- narzędzia do wizualizacji poszczególnych operacji,
- narzędzia umożliwiające dokonanie pełnej rozgrywki na bazie misji.
- narzędzia umożliwiającego tworzenie światów,

do organizacji projektu zostało wykorzystanie podejście modularnego monolitu. Organizacja kodu w ramach modularnego monolitu składa się na taką organizację kodu, kiedy w jednym repozytorium znajduje się wiele usług odpowiadających za różne części systemu. Takie podejście umożliwia łatwiejsze wdrożenia poszczególnych usług na różne sposoby – jako końcówki REST API czy jako pakiety niezbędne do działania innych usług. Modularny monolit jest określany jako rozwiązanie pośrednie pomiędzy klasycznym monolitem a architekturą opartą o mikroserwisy.

Kończówki REST API zostały wdrożone na platformie chmurowej Azure z wykorzystaniem Azure Functions i, jak reszta projektu, zostały napisane w języku Python 3.9. Komunikacja z końcówkami odbywa się za pomocą protokołu HTTP. Użytkownik wysyła żądanie do odpowiedniej końcówki i otrzymuje odpowiedź. Aby ułatwić użytkownikom korzystanie ze wszystkich końcówek, został przygotowany specjalny zestaw żądań, wraz z przykładowymi parametrami, udostępniony następnie przy pomocy narzędzia Postman⁸.

Technologie wykorzystane w implementacji referencyjnej

JSON Schema

JSON Schema⁹ jest narzędziem, za pomocą którego możemy określić ograniczenia struktury danych zapisanych w formacie JSON mocniejsze niż wynikające z samej definicji formatu. Możemy określić z jakich atrybutów mają składać się obiekty, jakie elementy możemy przechowywać w tablicach, definiować ich licznosc czy wymagalność. Możemy również tworzyć „podschematy”, przypominające definicję encji, które później możemy wykorzystywać wielokrotnie w głównym schemacie. Jest to bardzo wygodne przy definiowaniu zagnieżdżonej struktury grafowej składającej się z węzłów o analogicznej konstrukcji.

Podstawowym sposobem budowy schematu jest wskazanie głównego węzła dokumentu i opisanie par *atrybut: wartość* pod kilkoma względami:

- jakiego typu lub typów wartości możemy przechowywać w danym atrybucie lub jakie konkretne wartości danego typu możemy przechowywać
- które atrybuty są wymagane
- możemy uzależniać wymagalność jednych atrybutów od występowania innych
- czy mogą pojawiać się atrybuty niezdefiniowane w schemacie.

⁸ <https://www.postman.com/red-space-697620/workspace/ztg-uj/collection>

⁹ <https://json-schema.org/>

W przykładzie kodu 9 pokazany jest przykładowy schemat z typowymi elementami. Używając atrybutu `$ref` możemy wskazać, że elementami tablicy `Locations` mogą być obiekty `node` zdefiniowane wcześniej w schemacie.

Kod 9. Przykładowy schemat JSON Schema.

```
{
  "$schema": "http://json-schema.org/draft-07/schema",
  "$id": "http://www.ztg.fais.uj.edu.pl/",
  "title": "StoryGraph",
  "description": "JSON Schema for productions.",
  "type": "array",
  "definitions": {
    "node": {...}
  },
  "items": {
    "type": "object",
    "properties": {
      "Title": {"type": "string"},
      "Override": {"type": "integer", "enum": [0, 1, 2]},
      "LSide": {
        "type": "object",
        "properties": {
          "Locations": {
            "type": "array",
            "items": {
              "anyOf": [{"$ref": "#/definitions/node"}]
            }
          }
        }
      },
      "additionalProperties": false,
      "required": ["Locations"]
    }
  },
  "additionalProperties": false,
  "required": ["Title", "LSide", "Override"]
}
```

REST API

REST API (ang. *representational state transfer + application programming interface*) – jest rozwiązaniem architektonicznym, wykorzystywanym przy projektowaniu systemu lub usługi w oparciu o bezstanowy protokół HTTP. Jest to niezwykle elastyczne podejście (w odróżnieniu od SOAP czy XML-RPC), ponieważ nie narzuca ścisłej struktury. Do opracowywania tego rodzaju interfejsów możemy użyć wielu języków programowania – przede wszystkim takich jak Python, Java, C# czy PHP, które umożliwiają łatwą integrację napisanego kodu z serwerem WWW. Format wymiany danych również nie jest narzucony – możemy korzystać z dowolnego formatu jak np. JSON, XML czy YAML. Założenia, jakie narzuca na nas REST API zawierają się w sześciu zasadach:

- jednolity interfejs – żądania wysyłane przez klientów związane z tym samym zasobem, powinny wyglądać tak samo, bez względu na źródło, z którego pochodzą,
- architektura klient-serwer – aplikacja kliencka i aplikacja obsługująca żądania (serwer) muszą być niezależne od siebie,
- bezstanowość – każde żądanie wysyłane przez klienta musi zawierać wszystkie niezbędne informacje potrzebne serwerowi do przetworzenia takiego żądania, z uwzględnieniem danych identyfikujących danego klienta, jak token czy identyfikator sesji,
- warstwowa architektura systemu – aplikacja kliencka nie może zakładać, że łączy się bezpośrednio z serwerem, ze względu na to, że w takiej komunikacji mogą pojawić się dodatkowe systemy pośredniczące (ang. *proxy*) czy równoważące obciążenie (ang. *load balancer*),
- udostępnianie kodu – ta zasada jest opcjonalna, serwer może w odpowiedzi przesłać do klienta kod wykonywalny (np. aplet Javy).

Azure Functions

Azure Functions¹⁰ – jest rozwiązaniem dedykowanym dla chmury Azure, z kategorii obliczeń bezserwerowych (ang. *serverless computing*). Dzięki wykorzystaniu takiego rozwiązania nie ma potrzeby obsługi takich elementów jak system operacyjny czy sieć. Możemy wdrożyć kod, który będzie działał w ramach jednej funkcji, na platformie dostarczanej przez Microsoft Azure. W projekcie StoryGraph, Azure Functions zostały wykorzystane do przygotowania REST API. Aby udostępnić użytkownikom możliwość wizualizacji danych przy pomocy GraphViz został przygotowany specjalny obraz, który umożliwiał instalację wymaganych bibliotek w środowisku Azure Functions. Obraz jest budowany przy pomocy narzędzia Docker na platformę Linux z wykorzystaniem dedykowanego pliku zawartego w projekcie – Dockerfile, który opisuje sposób tworzenia obrazu, instalację bibliotek i uruchomienie końcówek REST API.

¹⁰ <https://docs.microsoft.com/en-us/azure/azure-functions/functions-overview>

Aby ułatwić użytkownikom korzystanie z końcówek REST API, w ramach projektu została przygotowana kolekcja końcówek REST API, wraz z przykładowymi żadaniami, wskazującymi ich strukturę. Za pomocą narzędzia Postman można wysyłać i odbierać zapytania do końcówek REST API opartych na protokole HTTP wraz z możliwością pełnej modyfikacji parametrów żądania i podglądu odpowiedzi.

Biblioteka GraphViz 3

Do wizualizacji grafów została wykorzystana biblioteka GraphViz 3 bazująca na silniku DOT. Możliwości wizualizacji obejmują nie tylko GWSG ale także lewe i hipotetyczne prawe strony poszczególnych produkcji, warianty dopasowania lewej strony do świata oraz hierarchię produkcji. Są one dostępne zarówno w projekcie, dla osób, które posiadają dostęp do kodu źródłowego, jak również częściowo w sposób otwarty dla użytkowników poprzez końcówki REST API.

Aplikacja konsolowa StoryGraph

Dostępne pakiety i moduły

- **api_examples, api_json_validation, api_visualise_production**
 - Azure Function do udostępniania schematu poprzez REST API
 - Azure Function do walidowania plików produkcji i świata poprzez REST API
 - Azure Function do generowania schematów produkcji poprzez REST API
- **json_validation**
 - json_validate: do walidowania plików produkcji i świata lokalnie
- **manual_world_modifications**
 - world_modifications: do ręcznego modyfikowania wskazanego pliku świata przez projektanta
- **production_hierarchy (pomocnicze)**
 - visualise_production_hierarchy: wyświetlanie hierarchii produkcji
- **production_match (pomocnicze)**
 - find_productions_to_perform: znajduje i wypisuje listę produkcji pasującą do wskazanego na początku skryptu świata. Szczegóły dopasowań wizualizuje w katalogu production_matches/out
- **production_processor**
 - application: Przeprowadza użytkownika przez proces decyzyjny gry zaczynając od stanu startowego wskazanego na początku skryptu świata. Znajduje dopasowania produkcji z plików misji wskazanych na początku skryptu, wykonuje wybraną

przez użytkownika produkcję w wybranym wariantcie dopasowania i od nowa znajduje dopasowania produkcji do zmienionego świata dopóki użytkownik nie przerwie cyklu. Zapisuje wizualizacje kolejnych stanów świata i warianty dopasowań w katalogu. Pozwala na zapisanie tamże aktualnego stanu świata w pliku .json.

5.4 Funkcjonalność implementacji referencyjnej

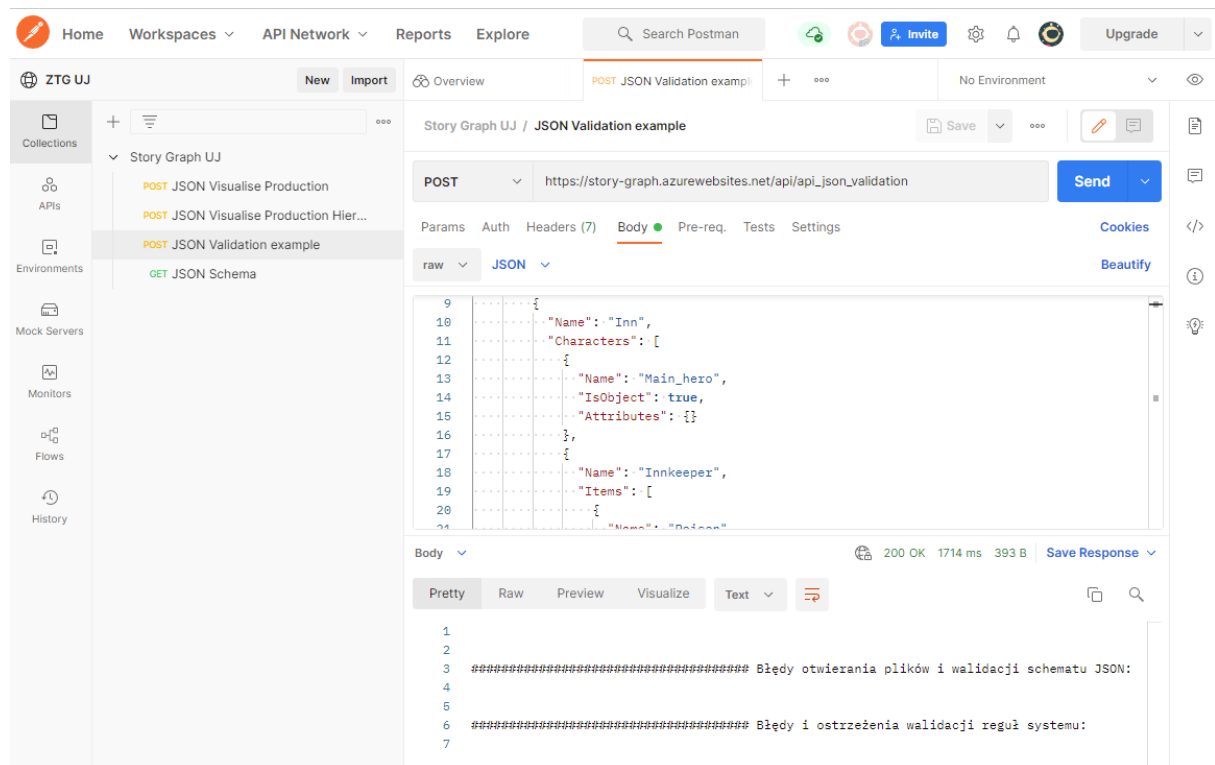
Walidacja danych

Aby móc weryfikować, czy stworzone produkcje są zgodne z przyjętym schematem, został przygotowany zestaw narzędzi weryfikujący ich poprawność. Bazowym jego elementem jest schemat produkcji, opisujący strukturę obiektu JSON, informacje o tym, które atrybuty są wymagane, a które opcjonalne, ich umiejscowienie w strukturze obiektu oraz informacje o dopuszczalnych wartościach, z uwzględnieniem rekurencyjnego zagnieżdżania poszczególnych obiektów wewnątrz innych, do dowolnego poziomu. Schemat został przygotowany w oparciu o standard JSON Schema (por. rozdział 5.3).

Drugim krokiem walidacji jest sprawdzenie, czy przygotowane produkcje zgodne są ze szczegółowymi wytycznymi specyfikacji, które ze względu na wzajemne zależności nie mogą być wyrażone bezpośrednio w schemacie.

Walidacja produkcji odbywa się w dwóch trybach: po pierwsze w osobnym skrypcie, którego celem jest wyświetlenie / umieszczenie w logu informacji o błędach i ostrzeżeniach, czyli konstrukcjach niepoprawnych lub podejrzanych: podobnych do typowych błędnych rozwiązań. W tym trybie walidacja dostępna jest zarówno bezpośrednio w projekcie w ramach odpowiedniego pakietu jak i poprzez końcówkę REST API. Interfejs graficzny walidatora pokazany jest na rysunku 22.

W drugim trybie skrypt walidujący nie generuje listy błędów ani ostrzeżeń, ale buduje zbiór poprawnych produkcji do dalszego wykorzystania w innych skryptach, np. w skrypcie symulacji rozgrywki. Pozwala to w kolejnych skryptach traktować dane wejściowe jako poprawne i wykonywać działania nie obawiając się, że błędy struktury zablokują działania skryptu.



Rys. 22. Przeglądarkowy interfejs użytkownika udostępniany przez walidator poprawności produkcji grafowych.

Testowanie linii fabularnych (symulacja rozgrywki)

Skrypt przeznaczony jest do testowania linii fabularnych. Pozwala wybrać jedną linię fabularną, prześledzić ją krok po kroku, zapisać w grafie rozgrywki i doprowadzić do jednego ze stanów końcowych gry. Nie jest to implementacja gry, między innymi dlatego, że pozwala ręcznie sterować postaciami niezależnymi pomiędzy ruchami gracza, aby spośród dostępnych dla nich działań wykonane zostały te oczekiwane przez użytkownika (nie nazywamy go w tym momencie graczem) a także nie ukrywa przed użytkownikiem efektu wybieranego działania. O różnicach między symulacją linii fabularnej a grą więcej jest napisane w rozdziale 6.3.

Inicjalizacja symulowanej rozgrywki polega na wskazaniu jednego pliku JSON zawierającego zdefiniowany świat i co najmniej jednego pliku JSON zawierającego dostępne produkcje. Zazwyczaj zamiast jednego są to trzy pliki: z uniwersalnymi produkcjami generycznymi, produkcjami wykonywanymi automatycznie po zaistnieniu warunków dopasowania oraz z produkcjami zaprojektowanymi specjalnie dla danej misji. Ostatnim krokiem inicjalizacji jest wskazanie głównego bohatera – postaci, którą będziemy sterować z pozycji gracza.

Kod 10. Pierwszy ekran wyników wykonania konsolowego skryptu symulacji rozgrywki gracza w grze fabularnej.

```

SYMULATOR   PROCESU   DECYZYJNEGO   GRACZA   RPG

Proces decyzyjny fabuły zdefiniowanej w świecie: World_q00
poprzez produkcje generyczne i misję: quest00_Dragon_story
dla bohatera: Main_hero.
Wizualizacje kolejnych możliwości wyboru i wykonanych produkcji znajdują się w katalogu:
/mnt/d/GitHub/StoryGraphPython/gameplays/gp-20220419174539

Podaj nazwę gracza: IGG

Jesteś głównym bohaterem gry (Main_hero). Twoim celem w tej misji jest zdobycie
smoczego jaja pilnowanego przez groźnego smoka. Smoka możesz zabić w walce (pod
warunkiem, że uzyskasz odpowiednio dużo siły), otruć lub wypłoszyć z legowiska.
Siłę zwiększamy jedząc obiekty o dużej wartości odżywczej, truciznę dostaniemy
od przyjaciół lub pozyskamy z trujących roślin.

Lokacja Road nie ma żadnych własności.
Przebywa w niej:
⊙ Main_hero {'HP': 100, 'Money': 10}
   który posiada przedmioty:
   □ Sword {'Value': 1000}

#### Co może zrobić Main_hero:
Z 46 produkcji udało się dopasować 5.
00. Teleportation - 7 wariantów
01. Location change - 5 wariantów
02. Dropping item - 1 wariantów
03. Deleting the group - 1 wariantów
04. Character's death - 1 wariantów

0-4 - wybór produkcji, enter - opuszczenie kolejki, „end” - koniec symulacji, „save” - zapis świata
Co robi Main_hero? 1

#### Produkcja „Location change” ma 5 wariantów.
#### Jeżeli chcesz poznać szczegóły wariantów, wygeneruj wizualizacje (katalog podany u góry).
00. Lokacja_A = Road, Bohater = Main_hero, Lokacja_B = Wizards_hut,
01. Lokacja_A = Road, Bohater = Main_hero, Lokacja_B = Forest,
02. Lokacja_A = Road, Bohater = Main_hero, Lokacja_B = Pasture,
03. Lokacja_A = Road, Bohater = Main_hero, Lokacja_B = Village,
04. Lokacja_A = Road, Bohater = Main_hero, Lokacja_B = Inn,

0-4 - wybór wariantu, enter - opuszczenie kolejki, d - wizualizacja wariantów
Co konkretnie robi Main_hero w produkcji?

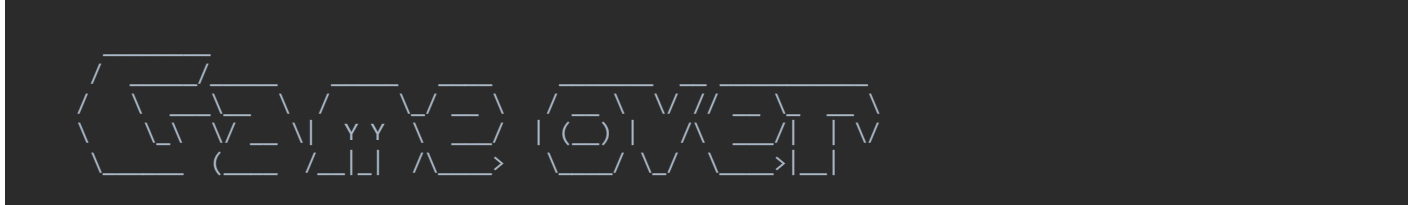
```

Po procedurze inicjalizacji cała rozgrywka odbywa się w głównej pętli skryptu i toczy się nieprzerwanie aż do osiągnięcia stanu świata z dwóch kategorii: stanu odpowiadającego warunkowi (warunkom) wygranej lub śmierci głównego bohatera. Przykładowy ekran otwierający działanie skryptu pokazany jest w przykładzie kodu 10 a ekran zamykający działanie skryptu

w dwóch wariantach przedstawiony jest w przykładach kodu 11 i 12. Schematyczny zapis struktury pętli znajduje się w przykładzie kodu 13.

Kod 11. Ostatni ekran wyników wykonania skryptu po osiągnięciu warunku końcowego – śmierci bohatera.

Nie ma postaci Main_hero w świecie. Zniknął główny bohater po ruchu NPC-a. Pewno zginął.



Kod 12. Ostatni ekran wyników wykonania skryptu po osiągnięciu warunku końcowego – wygranej.



Ponieważ opisywana implementacja obejmuje grę turową, ruchy postaci odbywają się naprzemiennie: ruch gracza, akcje automatyczne świata, ruchy postaci niezależnych, akcje automatyczne świata. Działanie algorytmów można prześledzić na przykładzie ruchu gracza.

Pojedyncza tura odbywa się w następującym porządku:

- Znalezienie wszystkich podgrafów GWSG będących w relacji dopasowania generycznego z lewą stroną którejkolwiek produkcji (grafu czterowarstwowego z częściowym bądź pełnym etykietowaniem).

- Wykonanie procedury sprawdzania predykatów stosowalności na każdym ze znalezionych podgrafów i odrzucenie podgrafów niespełniających jej.
- Przedstawienie użytkownikowi informacji o dopasowanych produkcjach we wszystkich wariantach (por. przykład kodu 10).
- Wykonanie, z argumentami w postaci węzłów z wariantu wskazanego przez gracza, procedury modyfikującej przekształcany graf wyrażonej listą instrukcji związanej z produkcją wskazaną przez gracza.

Kod 13. Pseudokod przedstawiający strukturę algorytmu symulowania rozgrywki.

```
main_loop():
    while (is_victory_condition == false or player_character_alive == true):
        mathing_productions = production_list
        for production in production_list:
            if is_matching_world( production ) == false:
                mathing_productions.remove(production)
                continue
            if is_precondition_valid( production ) == false:
                mathing_productions.remove(production)

        action = user_select_availible_action( mathing_productions )
        apply( action )
```

Serce skryptu, czyli algorytm znajdowania dopasowania generycznego opisany jest w rozdziale 5.1.

Modelowanie świata

Z punktu widzenia projektanta istotnym elementem pracy nad grą jest przygotowanie świata, w którym produkcje mają być wykonywane. Modyfikowanie świata nie bezpośrednio w treści pliku JSON, lecz poprzez skrypt (por. kod 14), umożliwia uniknięcie pomyłek w strukturze pliku i przyspiesza modyfikacje.

Kod 14. Efekt uruchomienia konsolowego skryptu modyfikacji GWSG.

```

M O D Y F I K A C J A   S T R U K T U R Y   Ś W I A T A

Modyfikacja świata pobranego z pliku: World_PWK2020_base
Wizualizacje kolejnych możliwości wyboru i wykonanych produkcji znajdują się w katalogu:
/mnt/d/Uniwersytet Jagielloński/StoryGraph - General/materiały fabularne/productions/worlds.
Tam też zostanie zapisany końcowy plik JSON i odpowiadający m PNG.
Pliki pomocnicze (wizualizacje każdej zmiany) znajdują się w katalogu:
tamże w podkatalogu: World_PWK2020_base_20220420003517.

UWAGA: Nazwy postaci i lokacji zdefiniowane są w katalogu:
/mnt/d/GitHub/StoryGraphPython/json_validation/allowed_names.

### Co może zrobić projektant ze światem: World_PWK2020_base
00. Usuń niepotrzebne węzły
01. Dodaj węzeł
02. Usuń węzeł
03. Przenieś węzeł
04. Zmień nazwę węzła
05. Zmień atrybuty węzła
06. Dodaj połączenie między lokacjami
07. Usuń połączenie między lokacjami
08. Dodaj obiekty z misji (jeszcze nie działa)
save - zapisz świat
end - zakończ

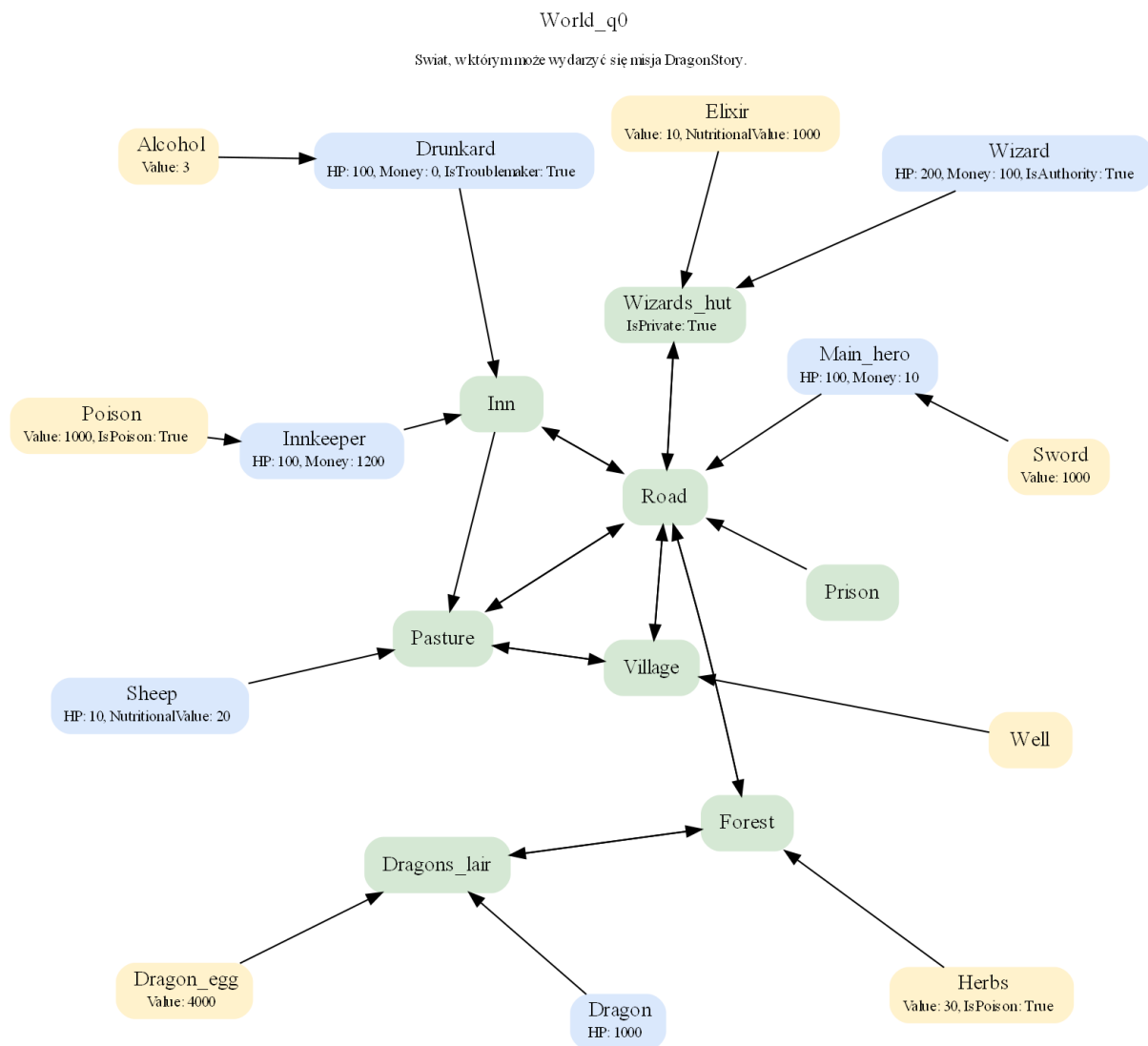
0 - usuń wiele, 1 - dodaj, 2 - usuń, 3 - przenieś, 4 - zmień nazwę, 5 - zmień atrybuty
6 - dodaj połączenie, 7 - usuń połączenie, 8 - dodaj obiekty z misji, 9 - uzupełnij wymagane atrybuty,
save, end
Które działanie wybierasz:

```

Wizualizacja świata

Funkcją pomocniczą wykorzystywaną przez projektantów i testerów jest możliwość wygenerowania obrazu grafu świata. Obraz w postaci pliku PNG lub PDF generowany jest za pomocą biblioteki GraphViz przy użyciu silnika „neato”. Węzły poszczególnych warstw różną się kolorem (zielony dla warstwy lokacji, niebieski dla postaci, żółty dla przedmiotów a fioletowy dla informacji fabularnych). Parametry wizualizacji pozwalają umieścić w węźle jego etykietę, opcjonalnie identyfikator (łańcuch tekstowy pozwalający odróżnić węzły o identycznych etykietach) oraz listę atrybutów wraz z wartościami.

Rozmieszczenie węzłów odbywa się automatycznie w procedurach biblioteki GraphViz i na przykładzie na rysunku 23 możemy zauważyć, że nawet dla grafów planarnych nie zawsze wybierane jest rozłożenie optymalne pod względem nieprzecinania się krawędzi.



Rys. 23. Przykładowy graf świata.

Wizualizacja produkcji

Możliwość wizualnego przedstawienia produkcji jest kolejną funkcją pomocniczą przygotowaną dla projektantów. Przedstawienie takie musi zawierać obraz przedstawiający podgraf, którego dopasowania poszukujemy (lewą stronę produkcji) oraz graficzne przedstawienie zmian wykonywanych w grafie po dopasowaniu, ale bez uwzględnienia kontekstu, którego nie znamy.

Lewa strona produkcji wizualizowana jest analogicznie do wizualizacji świata, ale z użyciem silnika „dot”, który lepiej nadaje się do przedstawiania grafów o strukturze drzewiastej, czyli pojedynczych snopków. Ponieważ lewa strona produkcji dopuszcza częściowe etykietowanie, to niektóre węzły nie będą opisane lub będą odróżniane wyłącznie na podstawie identyfikatora.

Wskazane są nazwy atrybutów wymaganych w lewej stronie, możliwe jest także wskazanie parametrem, czy chcemy, aby proste predykaty stosowalności wymagające konkretnej wartości atrybutu wizualizowały się tak, jak wartości atrybutów w instancji grafu.

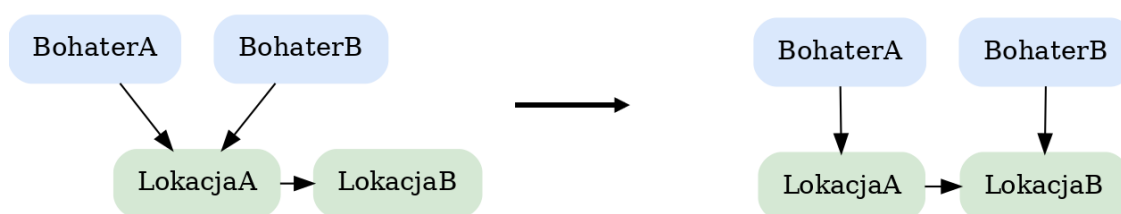
Sposób wizualizacji zmian zależy od tego, jakiego typu instrukcje wykorzystywane są w produkcji (por. rozdział 4.4, podrozdział *Typy instrukcji*).

Jeśli modyfikacje ograniczone są do węzłów dopasowanych do lewej strony, to przedstawienie graficzne zmian może być wzorowane na standardowym układzie wizualizacji klasycznie zdefiniowanej produkcji grafowej z użyciem prawej strony, gdzie prawa strona wskazuje, jakich dokonano manipulacji węzłami dopasowanymi w lewej stronie produkcji a podgrafy lewej i prawej strony rozdzielone są strzałką.

Drugi poziom zależności węzłów modyfikowanych od węzłów dopasowanych pozwala na modyfikacje węzłów z całego snopka węzła dopasowanego do lewej strony produkcji. Wizualizację takiej produkcji możemy stworzyć generując obraz analogiczny do opisanego w poprzednim akapicie, ale tylko dla zmian węzłów dopasowanych a następnie dodać do niego pseudowęzły mające zasignalizować potencjalne węzły ze snopka ulegające przekształceniom. Zaproponowany kod wizualny pozwala zaznaczyć możliwe manipulacje węzłami ze snopka, ale nie wskazuje jednoznacznie ich zakresu.

Wizualizacja produkcji zawierającej instrukcje trzeciego poziomu zależności nie została zaimplementowana.

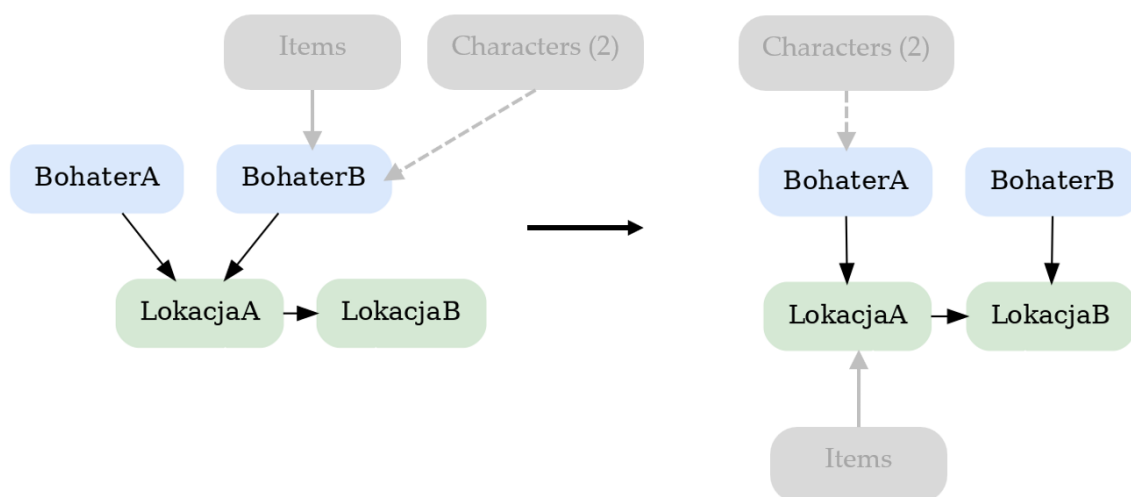
Aby przedstawić różnice między wizualizacjami produkcji zawierających instrukcje pierwszego i drugiego poziomu zależności rozważmy hipotetyczną produkcję, w której dwie postaci podejmują interakcję, zakończoną dla jednej z nich udaniem się do sąsiedniej lokacji. W tak zdefiniowanym działaniu nie ma potrzeby wykorzystywania multireferencji, da się jednoznacznie określić prawą stronę produkcji, której wizualizacja pokazana jest na rysunku 24.



Rys. 24. Wizualizacja produkcji interakcji zakończonej zmianą lokacji. Wizualizacja lewej strony produkcji i wizualizacja manipulacji na węzłach wzorowana na wyglądzie prawej strony produkcji.

Możemy jednak zdefiniować tę interakcję bardziej szczegółowo, jako przegnanie z przejęciem zasobów i dodać do niej działanie polegające na tym, że podczas ucieczki przegrany bohater gubi wszystkie przedmioty a co najwyżej dwie postaci, nad którymi sprawuje kontrolę (np. wcześniej pojmani jeńcy) przechodzą pod kuratelę bohatera wygrywającego. Wtedy wizualizacja z rysunku 24, która pokazuje, co dzieje się z węzłami dopasowanymi do lewej strony produkcji, nie oddaje całości zmian w świecie. Możemy zamodelować je tak, jak pokazano na rysunku 25, ale musimy pamiętać, że jest to wizualizacja pogładowa i nie wiemy, czy przegrywająca postać miała jakichkolwiek jeńców lub jakiegokolwiek przedmioty. Jeśli postać nie posiada żadnych zasobów, czy to przedmiotów, czy podległych postaci, to produkcja i tak zostanie wykonana, bo działania wskazane przez multireferencje to działania warunkowe – wykonają się, jeśli liczba znalezionych obiektów jest niezerowa, ale ich brak nie blokuje wykonania produkcji.

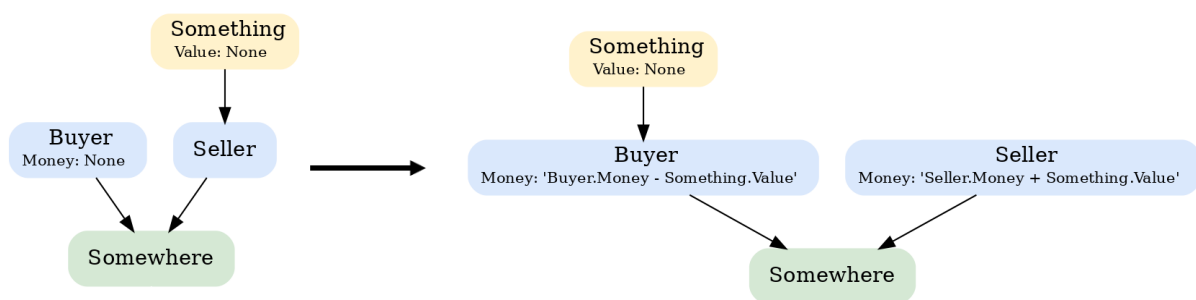
Wykorzystana na poniższych rysunkach konwencja wizualna jest następująca: obiekty wskazane multireferencją i krawędzie z nich wychodzące rysujemy kolorem szarym. Jeśli multireferencja kończyła się symbolem „*”, to zamiast etykiety umieszczamy nazwę warstwy z ewentualnym wskazaniem ograniczenia liczby elementów w nawiasie. Jeśli taki opis węzłów jest niejednoznaczny, dodajemy identyfikatory. Jeśli multireferencja zawiera symbol „**”, krawędź rysujemy linią przerywaną.



Rys. 25. Wizualizacja produkcji walki z ucieczką. Wizualizacja lewej strony produkcji i wizualizacja manipulacji na węzłach z uwzględnieniem działań związanych z obiektami wskazanymi poprzez multireferencje.

Od strony implementacyjnej generowanie prawej strony produkcji opiera się na omówionych w poprzednim rozdziale strukturach wynikowych algorytmu dopasowania produkcji. Jako

świat definiujemy kopię lewej strony produkcji, ale nie przeprowadzamy procedury dopasowania tylko przyjmujemy, że dopasowaniem dla każdego węzła jest jego odpowiednik w kopii. Dzięki temu omijamy niezgodności z modelem związane z tym, że od świata wymagamy pełnego etykietowania i określonych wartości atrybutów (ponieważ jest instancją). Następnie wykonujemy kolejne instrukcje. Instrukcje dotyczące węzłów ze snopków zostałyby zignorowane, ponieważ w świecie stworzonym z lewej strony nie ma żadnych węzłów poza bezpośrednio wskazanymi, dlatego warunkowe wykonanie tych instrukcji będziemy symulować dodając w wizualizacji sztuczne węzły o etykietach zbudowanych na podstawie multireferencji. Nowe wartości atrybutów (wizualizacja produkcji w celach poglądowych zawsze obejmuje tryb pokazywania wartości atrybutów) są w prawej stronie podawane *explicite*, jeśli w instrukcjach narzucane były jako stałe, lub zamieniane są na łańcuch tekstowy odpowiadający wyrażeniu, którym są wyliczane.



Rys. 26. Wizualizacja produkcji z wartościami wyliczonymi wyrażeniem.

Generowanie prawej strony produkcji

Między podgrafem prawej strony produkcji i zbiorem instrukcji zachodzi relacja pozwalająca odtworzyć podgraf prawej strony produkcji na podstawie instrukcji, ale wyłącznie w odniesieniu do węzłów bezpośrednio podanych w lewej stronie produkcji. Węzły ze snopka wskazane przez multireferencje nie mogą zostać uwzględnione w zapisie prawej strony ze względu na ograniczenia takiej formuły przedstawiania produkcji. Z tego względu w strukturze SHEAF-PRODUCTION zrezygnowano z podawania jawnie prawej strony produkcji.

Dla produkcji zawierających instrukcje operujące wyłącznie na węzłach z lewej strony produkcji możliwe jest przekształcenie klucza `Instructions` w dodatkowy nieuwzględniony w podstawowej specyfikacji klucz `RSide`. Ponieważ jest to działanie możliwe do wykonania tylko dla pewnego podzbioru produkcji a informacje z tak utworzonej prawej strony produkcji

są redundantne do informacji z instrukcji, klucz ten nie jest ani wymagany ani wykorzystywany w implementacjach.

Przykład takiego przekształcenia pokazują próbki kodu 15 i 16.

Kod 15. Definicja produkcji odkładania przedmiotu jako lewa strona produkcji + instrukcje.

```
{
  "Title": "Dropping item / Odłożenie przedmiotu",
  "TitleGeneric": "",
  "Description": "«Bohater» odkłada przedmiot («Something») do lokacji.",
  "Override": 0,
  "LSide": {
    "Locations": [
      {
        "Id": "LokacjaA",
        "Characters": [
          {
            "Id": "Bohater",
            "IsObject": true,
            "Items": [
              {
                "Id": "Something",
              }
            ]
          }
        ],
      }
    ],
  },
},
"Instructions": [
  {
    "Op": "move",
    "Nodes": "Something",
    "To": "LokacjaA/Items"
  }
]
}
```

Kod 16. Ta sama produkcja zdefiniowana jako lewa + prawa strona produkcji.

```
{
  "Title": "Dropping item / Odłożenie przedmiotu",
  "TitleGeneric": "",
  "Description": "«Bohater» odkłada przedmiot («Something») do lokacji.",
  "Override": 0,
  "LSide": {
    "Locations": [
      {
        "Id": "LokacjaA",
        "Characters": [
          {
            "Id": "Bohater",
            "IsObject": true,
            "Items": [
              {
                "Id": "Something",
              }
            ]
          }
        ]
      }
    ],
  },
  "RSide": {
    "Locations": [
      {
        "Id": "LokacjaA",
        "Characters": [
          {
            "Id": "Bohater",
            "IsObject": true,
            "Items": []
          }
        ],
        "Items": [
          {
            "Id": "Something",
          }
        ]
      }
    ]
  }
}
```

```

    }
  ]
}
]
}

```

Wizualizacja drzewa hierarchii produkcji

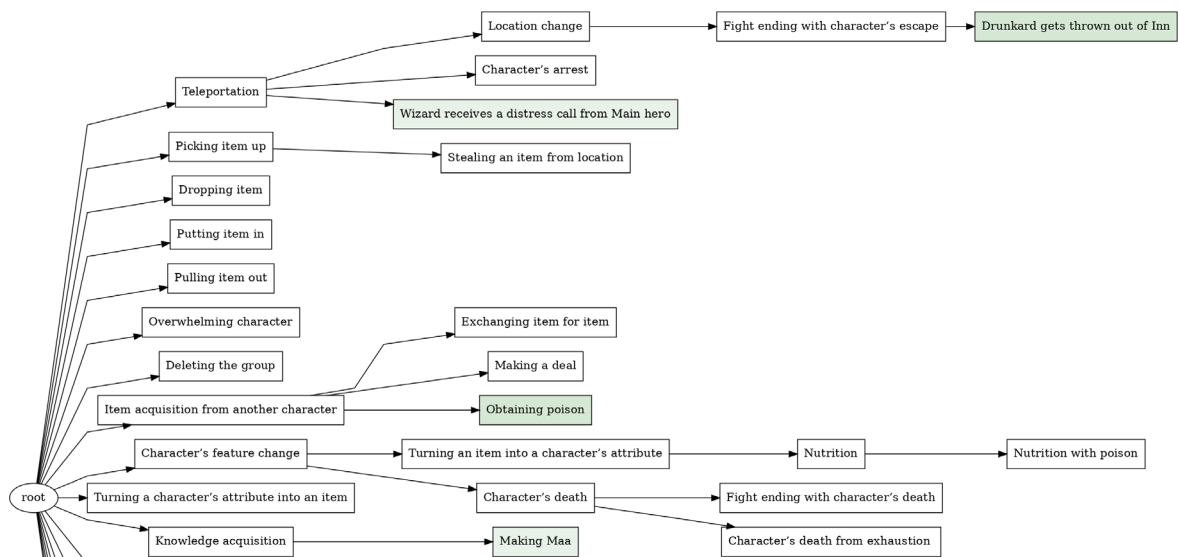
W trosce o spójność zachowań postaci w akcjach bardziej złożonych, ale bazujących na prostszych działaniach wprowadzono w systemie mechanizm dziedziczenia produkcji. O samej koncepcji i jej wpływie na mechaniki świata więcej napisane jest w rozdziale 6.1, podrozdział *Hierarchizacja produkcji*.

Produkcja bazowa dla danej produkcji określana jest przez projektanta w kluczu `TitleGeneric`. Nie może to być działanie zautomatyzowane, ponieważ wskazanie produkcji musi być świadomą decyzją projektową mającą konsekwencje przy wykonywaniu działań w trakcie rozgrywki, w szczególności niekiedy akcje bardziej szczegółowe obejmują działania pokrywające się z kilkoma produkcjami generycznymi i projektant musi wybrać, która z nich jest bazowa dla tworzonej właśnie produkcji szczegółowej. Będzie to istotne dla określania procedury przysłaniania produkcji, która jest opisana w rozdziale 6.1, podrozdział *Przesłanianie produkcji*.

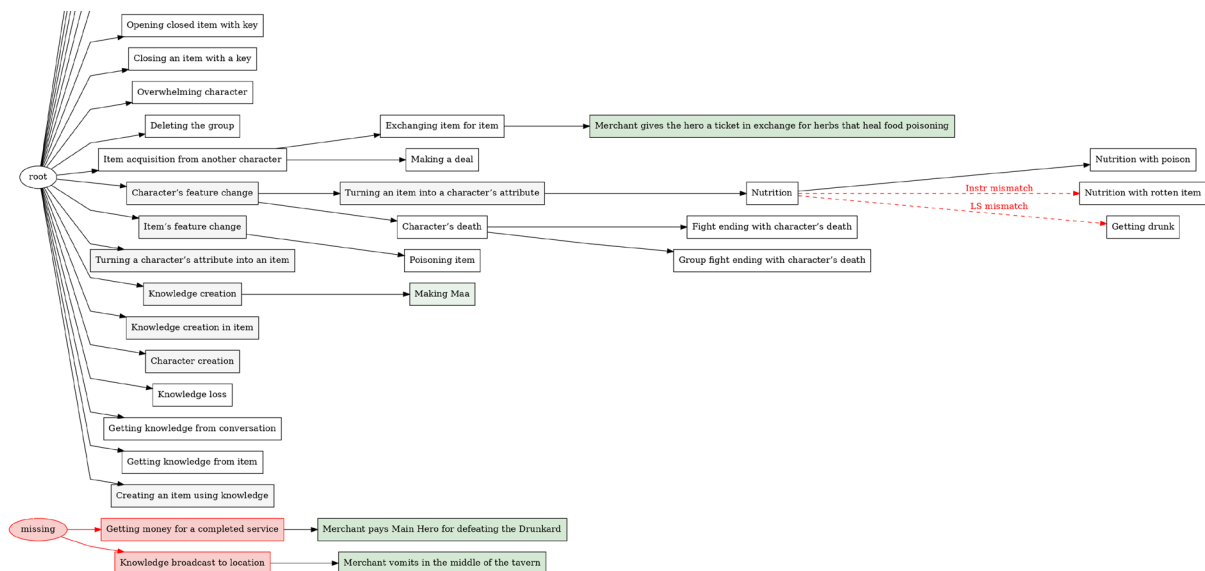
Budowanie drzewa hierarchii produkcji obejmuje więc etap walidacji, w którym sprawdzana jest deklaracja projektanta dotycząca dziedziczenia i wyświetlane ewentualne błędy. Docelowo w systemie wspomagającym tworzenie produkcji będzie możliwość wyboru produkcji bazowej w kreatorze produkcji i uzupełnienie jej o elementy dodatkowe bez możliwości modyfikacji wierzchołków i krawędzi bazowych inne niż dodanie etykiet i atrybutów, co zapobiegnie powstawaniu błędów. W momencie składania niniejszej pracy ta funkcja systemu jest jeszcze w fazie testów, więc dostępna jest wyłącznie usługa walidacji i wizualizacji drzewa produkcji. Usługa walidacji będzie potrzebna nawet w momencie udostępnienia generatora produkcji, ponieważ nie możemy wykluczyć ręcznej modyfikacji plików źródłowych poza systemem.

Wizualizacja hierarchii produkcji dostępna jest zarówno bezpośrednio w projekcie w ramach odpowiedniego pakietu jak i poprzez końcówkę REST API.

Wizualizacja drzewa hierarchii produkcji i oznaczania możliwych błędów przedstawiona jest na rysunkach 27 i 28.



Rys. 27. Przykładowa hierarchia produkcji.



Rys. 28. Przykładowa hierarchia produkcji z zaznaczonymi błędami.

6 Zastosowania systemu

Stworzenie przedstawionego w poprzednich rozdziałach modelu grafowego fabularnej gry przygodowej nie było celem samym w sobie. Impulsem do jego powstania była chęć kontrolowania spójności gry i wynikająca z niej potrzeba formalizacji zależności między akcjami gracza. Ułatwia to proces projektowania gry dzięki zdefiniowaniu najmniejszych całości fabularnych i zależności między nimi oraz zaimplementowaniu narzędzi ułatwiających walidację, wizualizację i testowanie projektu.

6.1 Model grafowy a mechanika gry

Podczas projektowania możliwych działań w grze po pierwsze – wychodzimy zawsze od mechaniki gry, która umożliwia ale i ogranicza działania bohaterów, po drugie – musimy dbać o spójność i logikę wydarzeń w grze a po trzecie – zależy nam na optymalizacji pracy projektantów i programistów.

W grze fabularnej bardzo istotna jest spójność świata. Oznacza ona, że schematy zachowań wypraktykowane w jednej sytuacji powinny wpasowywać się w sytuacje analogiczne a gracz może wykorzystywać metody dedukcyjne do przewidywania rozwoju sytuacji. Logika świata gry może różnić się od logiki świata rzeczywistego, ale musi być odczuwalna przez gracza. Konsekwencja w projektowaniu akcji oznacza nie tylko lepszą immersję gracza, ale także oszczędności czasu programistów i testerów, jeśli pewne akcje da się zunifikować.

Produkcje generyczne i szczegółowe

Sztandarowym przykładem ujednolicenia działań postaci jest wykorzystanie koncepcji produkcji generycznych do definiowania najbardziej podstawowych działań wynikających bezpośrednio z mechanik niezależnie od podmiotu je wykonującego. Są one odpowiednikami rutynowych czynności, które mogą mieć potencjał fabularny, ale same w sobie nie stanowią zwrotu akcji. Przykładem będą: przechodzenie między lokacjami, handel, walka – które są typowym zajęciem większości bohaterów gier przygodowych, sterowanych przez gracza i nie.

Przyjrzyjmy się przykładowej akcji przejścia bohatera z lokacji do lokacji, co będzie jednym z najczęściej podejmowanych działań, zarówno przez gracza jak i przez postaci niezależne. Jeśli wykorzystamy mechanizm częściowego etykietowania i zaprojektujemy działania – na poziomie interakcji między obiektami, animacji i interfejsu – dla jednego hipotetycznego gracza i dowolnych dwóch lokacji, to nie będziemy musieli projektować osobnych akcji dla każdego gracza i dowolnej kombinacji lokacji. W przypadku gry przedstawionej w rozdziale 2 dawałoby to (dla 21 lokacji i 15 bohaterów) 630 osobnych produkcji po 33 linijki kodu każda.

Po co więc w ogóle definiować produkcje szczegółowe? (W sensie ogólnym mówiąc o produkcjach szczegółowych mamy na myśli te, które mają pełne etykietowanie, ale wykorzystując koncepcję hierarchii produkcji możemy powiedzieć, że produkcją bardziej szczegółową będzie produkcja pochodna w drzewie hierarchii). Wykorzystujemy je do definiowania produkcji węzłowych. Niektóre bowiem działania bohatera w grze są jednostkowe i niepowtarzalne, powinien je wykonać wyłącznie gracz i to po spełnieniu konkretnych warunków. Warto też zauważyć, że najczęściej są one szczególnymi przypadkami akcji bardziej ogólnych, ale dodatkowe efekty wykonania takiego działania w stosunku do produkcji pierwotnej mogą otwierać możliwość np. ukończenia misji.

Hierarchizacja produkcji

Najistotniejsza dla gracza jest dostępność produkcji, czyli możliwość ich wykonania w bieżących okolicznościach. Podstawowym kryterium ograniczającym ich dostępność jest wymóg dopasowania lewej strony produkcji do świata, co jest istotą modelu grafowego, ale tak wygenerowaną listę akcji możemy chcieć dodatkowo ograniczyć. Po pierwsze: zgodnie z logiką przyczynowo-skutkową. Samo istnienie możliwości wykonania pewnych akcji powinno wykluczać inne, np. skoro rozważamy akcję ucieczki z więzienia, to znaczy, że nie da się z niego tak po prostu wyjść. Po drugie: w celu optymalizacji procesu decyzyjnego gracza. Możemy

założyć, że propozycje wykonania niektórych produkcji będą pojawiały się kontekstowo. Intuicyjnie wyczuwamy, że między produkcjami, szczególnie wywodzącymi się z produkcji bardziej ogólnych, zachodzą pewne zależności.

O hierarchii produkcji pod kątem uszczegóławiania bardziej ogólnych działań pisał już Władimir Propp w swoich analizach. Przywołajmy wspomniany w rozdziale 1.4 przykład ogólnie zdefiniowanego odejścia (opuszczenia), bardziej szczegółowego odejścia rodziców i jeszcze bardziej doprecyzowanej ich śmierci. Propp tworzył hierarchię produkcji ręcznie, my możemy wspomóc się własnościami modelu grafowego, ale należy pamiętać, że budowanie hierarchii produkcji, jeśli ma zachować formę liniową, jak u Proppa, albo drzewiastą, nie jest działaniem mechanicznym, jest formą twórczego wyboru. Automatyzacja pozwala wyłącznie nakreślić ramy owego wyboru.

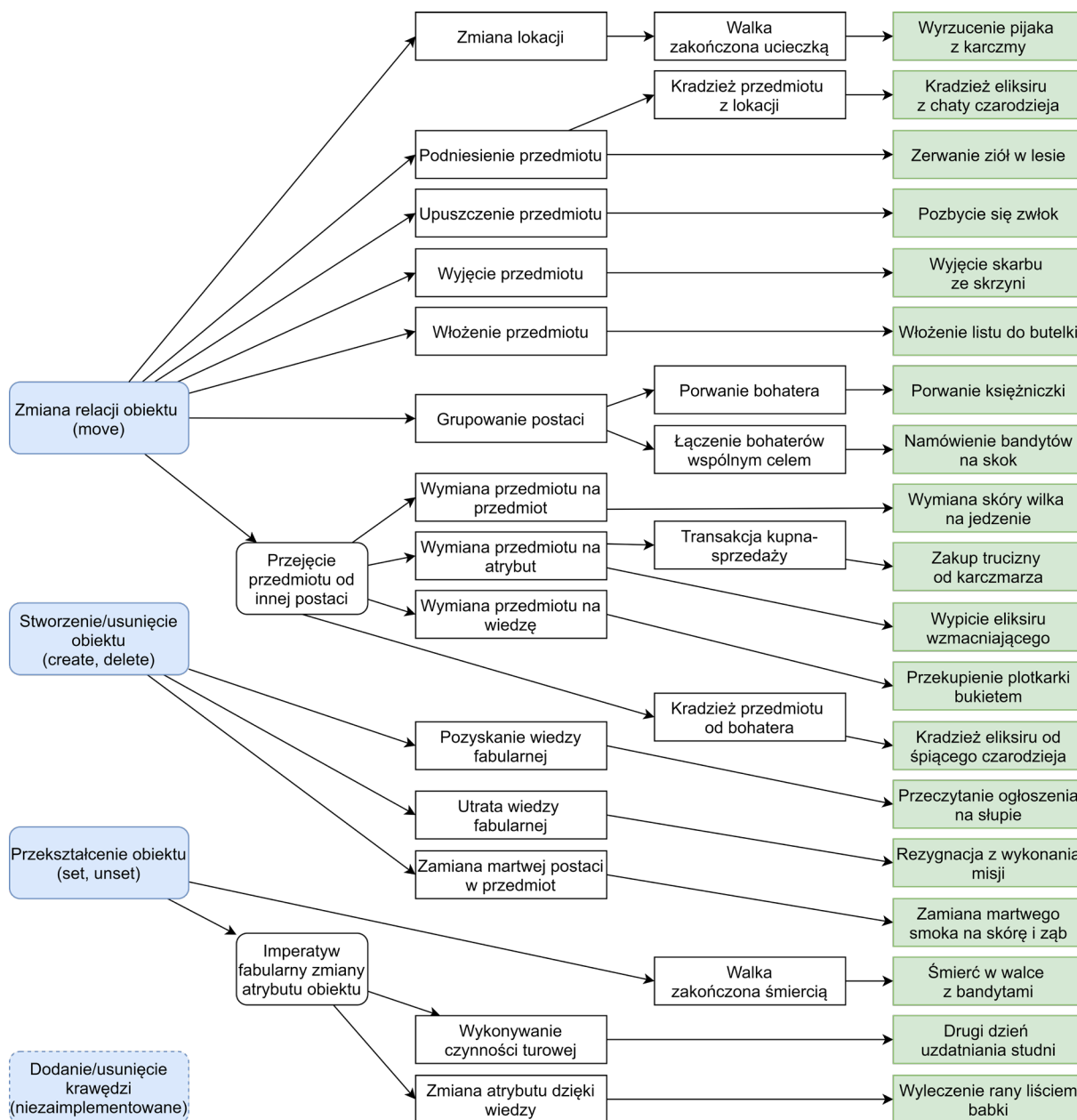
W przypadku modelu grafowego opisanego w niniejszej pracy wprowadzamy częściowe uporządkowanie produkcji korzystając z drzewa hierarchii produkcji i parametru `Override`. Zwróćmy uwagę, że wykorzystanie danej produkcji generycznej jako bazy do tworzenia produkcji pochodnych odbywa się na etapie projektowym i jest suwerenną decyzją projektantów gry. Zazwyczaj wiąże się ze zdefiniowaniem dodatkowych warunków lub czynności zmieniających znaczenie i konsekwencje działania w stosunku do bazowej produkcji.

Przykład tworzenia produkcji pochodnych: na podstawie produkcji generycznej „Przejęcie przedmiotu od innej postaci” definiujemy dwie produkcje: „Transakcja kupna-sprzedaży”, w której definiujemy zmniejszenie wartości atrybutu `Money` kupującego o wartość atrybutu `Value` przedmiotu a z kolei wartości atrybutu `Money` sprzedającego o tyle samo wzrasta, oraz „Kradzież przedmiotu”, w której atrybut `Money` dotychczasowego właściciela pozostawiamy bez zmian, złodzieja także, ale dodajemy mu węzeł z warstwy narracyjnej „Wystawienie za bohatera listu gończego”.

Z kolei na podstawie produkcji „Transakcja kupna-sprzedaży” możemy tworzyć kolejne pochodne produkcje, czy to kupowania przedmiotu z rabatem, czy to handlu substancjami nielegalnymi wymagającego znajomości dodatkowego hasła uzyskiwanego w osobnej misji.

Produkcje generyczne możemy uporządkować w hierarchę wywodząc działania coraz bardziej szczegółowe z produkcji bardziej ogólnych. Najbardziej ogólne produkcje (odpowiadające elementom mechaniki) nie będą bezpośrednio dostępne w grze, są stworzone wyłącznie jako punkt wyjścia do tworzenia produkcji bardziej szczegółowych (por. rys. 29).

W zapisie JSON produkcje generyczne odróżnia od produkcji szczegółowych zastosowanie parametru `Id` zamiast `Name` w opisie poszczególnych węzłów. Produkcja, w której każdy węzeł jest opisany przez parametr `Name`, jest z definicji produkcją szczegółową.

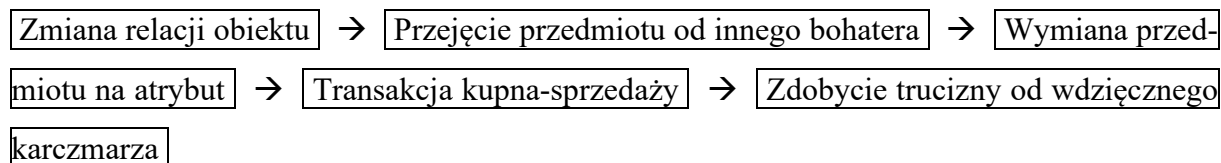


Rys. 29. Zestaw przykładowych produkcji wraz z zaznaczeniem ich poziomu w hierarchii. W owalnych ramkach produkcje hipotetyczne, niedostępne w grze, porządkujące hierarchię. W prostokątnych ramkach produkcje generyczne dostępne bezpośrednio w grze. Na niebiesko produkcje najwyższego poziomu, odzwierciedlające mechanikę. Na zielono produkcje najniższego poziomu (produkcje szczegółowe dotyczące głównego bohatera).

Rozwińmy drugi przykład: wyobraźmy sobie scenę, w której główny bohater kupuje od karczmarza truciznę. Jest to nielegalny produkt, więc karczmarz musi mieć specjalną motywację, żeby nam ten artefakt sprzedać. Zapewniamy to sobie dzięki zdobyciu jego wdzięczności w którejś z poprzednio wykonanych produkcji. Jest to transakcja kupna-sprzedaży, więc musi rządzić się tą samą logiką biznesową co kupowanie marchewki na targu. W sensie bardziej ogólnym wychodzimy od przejęcia przedmiotu od innej postaci, czyli przekazania obiektu

z ekwipunku (ang. *inventory*) jednego bohatera do ekwipunku drugiego, co na najbardziej ogólnym poziomie oznacza przeniesienie (operacją **move**) węzła z warstwy „Items” z jednego miejsca grafu do drugiego.

Hierarchia produkcji będzie wyglądała następująco:



Należy zwrócić uwagę, że moglibyśmy wydłużyć ten ciąg dodając produkcję: **Zdobywanie trucizny od wdzięcznego NPC-a** jako przedostatnią, gdyby specyfika gry tego wymagała.

Przeanalizujmy na powyższym przykładzie różnice w zapisie produkcji w formacie JSON: przykład kodu 17 przedstawia zapis produkcji generycznej „Transakcja kupna-sprzedaży” a przykład kodu 18 – zapis produkcji szczegółowej „Zdobywanie trucizny od wdzięcznego karczmarza”. Podstawową różnicą jest wykorzystanie w lewej stronie produkcji co najmniej jednego parametru **Id** w pierwszym przypadku a wyłącznie parametrów **Name** w drugim. Co więcej, zauważmy, że w produkcji szczegółowej pojawił się dodatkowy węzeł z warstwy narracji, dodatkowy predykat stosowalności i dodatkowe instrukcje. Gdyby nie było tych dodatkowych elementów to stworzenie produkcji szczegółowej przez projektanta byłoby technicznie możliwe, ale nie miałoby sensu, ponieważ stanowiłoby tylko ręczne uszczegółowienie produkcji generycznej, co może zostać wykonane automatycznie przez silnik fabularny.

Zwróćmy uwagę, że w części **Instructions** nie ma różnicy między produkcjami generycznymi a szczegółowymi, operujemy tam na referencji do węzła i nie ma znaczenia, czy została ona wyliczona z wartości **Id** czy **Name**.

Kod 17. Produkcja generyczna „Transakcja kupna-sprzedaży”. Na żółto parametry **Id** dające elastyczność dopasowania do różnych węzłów GWSG.

```

{
  "Title": "Making a deal / Transakcja kupna-sprzedaży",
  "TitleGeneric": "Item acquisition from another character / Przejęcie przedmiotu od innej postaci",
  "Description": "«Seller» sprzedaje kupującemu («Buyer») przedmiot («Something»).",
  "Override": 0,
  "LSide": {
    "Locations": [

```

```
{
  "Id": "Somewhere",
  "Attributes": {},
  "Characters": [
    {
      "Id": "Buyer",
      "IsObject": true,
      "Attributes": {
        "Money": null
      }
    },
    {
      "Id": "Seller",
      "IsObject": true,
      "Attributes": {
        "Money": null
      }
    },
    "Items": [
      {
        "Id": "Something",
        "Attributes": {
          "Value": null
        }
      }
    ]
  ]
}

],
"Preconditions": [
  {
    "Cond": "Buyer.Money >= Something.Value"
  }
],
```

```

    "Instructions": [
      {
        "Op": "move",
        "Nodes": "Something",
        "To": "Buyer/Items"
      },
      {
        "Op": "set",
        "Attribute": "Buyer.Money",
        "Expr": "Buyer.Money - Something.Value"
      },
      {
        "Op": "set",
        "Attribute": "Seller.Money",
        "Expr": "Seller.Money + Something.Value"
      }
    ]
  }
}

```

Kod 18. Produkcja szczegółowa „Zdobycie trucizny od wdzięcznego karczmarza”. Na żółto parametry Name, które zastąpiły Id z produkcji generycznej oraz fragmenty produkcji, które zostały dodane do produkcji generycznej, czyniąc ją od niej istotnie różną.

```

{
  "Title": "Getting poison from the grateful Innkeeper / Zdobycie trucizny od
    wdzięcznego karczmarza",
  "TitleGeneric": "Making a deal / Transakcja kupna-sprzedaży",
  "Description": "Bohater wykorzystuje wdzięczność karczmarza i kupuje od niego truciznę.",
  "Override": 0,
  "LSide": {
    "Locations": [
      {
        "Name": "Inn",
        "Attributes": {},

```

```

    "Characters": [
      {
        "Name": "Main_hero",
        "IsObject": true,
        "Attributes": {
          "Money": null
        },
        "Narration": [
          {
            "Name": "InnkeepersGratitude",
            "Attributes": {
              "Strength": null
            }
          }
        ]
      },
      {
        "Name": "Innkeeper",
        "Attributes": {
          "Money": null
        },
        "Items": [
          {
            "Name": "Elixir",
            "Attributes": {
              "IsPoisoned": true,
              "Value": null
            }
          }
        ]
      }
    ]
  },
  ]
},
],
},

```

```

    "Preconditions": [
      {
        "Cond": "Main_hero.Money >= Elixir.Value"
      },
      {
        "Cond": "InnkeepersGratitude.Strength >= 1"
      }
    ],
    "Instructions": [
      {
        "Op": "move",
        "Nodes": "Elixir",
        "To": "Main_hero/Items"
      },
      {
        "Op": "set",
        "Attribute": "Main_hero.Money",
        "Expr": "Main_hero.Money - Elixir.Value"
      },
      {
        "Op": "set",
        "Attribute": "Innkeeper.Money",
        "Expr": "Innkeeper.Money + Elixir.Value"
      },
      {
        "Op": "delete",
        "Nodes": "InnkeepersGratitude"
      }
    ]
  }

```

Budowanie zbioru dostępnych produkcji

W czasie gry przed każdą decyzją gracza budowany jest zbiór dostępnych dla gracza produkcji. Są to produkcje dotyczące głównego bohatera (tylko takie gracz może świadomie inicjować). Po wybraniu przez gracza którejś opcji produkcja jest wykonywana, przebudowywany jest graf stanu świata, ewentualnie wykonywane są produkcje wynikające z diagramów

sterujących NPC-ów, generowana jest nowa lista dostępnych produkcji dla gracza i cykl się powtarza.

System, w którym tylko produkcje szczegółowe są dostępne dla gracza, powoduje, że gra jest zamknięta, tzn. możliwe są tylko interakcje między wskazanymi *explicite* elementami świata gry. Sprowadza się to do wyboru przez gracza jednej z kilku ścieżek zaprojektowanych przez twórcę historii.

System, w którym na podstawie produkcji generycznych umożliwiane są interakcje gracza z dowolnymi elementami świata, jest znacznie bardziej otwarty, ale zwiększa się niebezpieczeństwo, że gracz znajdzie drogę „na skróty”, albo możliwe działania będą niezgodne z fabułą lub zdrowym rozsądkiem. Np. swobodna zmiana lokacji zastosowana do wszystkich lokacji powoduje, że pojęcie lokacji „więzienie” traci sens. Niezbędne jest tutaj wykorzystanie hierarchii stosowania produkcji, tzn. zdefiniowanie warunków, przy których istnienie produkcji szczegółowych blokuje wykonanie produkcji generycznych. Wykorzystamy do tego parametr **Override** produkcji szczegółowej, który może przyjmować trzy wartości: 0, 1 lub 2.

Parametr **Override** równy 0 oznacza, że dopasowanie produkcji pochodnej nie wpływa na możliwość wyboru produkcji bazowej, czyli jeśli została dopasowana pochodna, to gracz ma do wyboru dwie: pochodną i bazową.

Parametr **Override** równy 1 oznacza, że dopasowanie produkcji pochodnej blokuje możliwość wyboru produkcji bazowej. Jeżeli jednak produkcja pochodna nie została dopasowana, to musimy sprawdzić, czy gracz może wykonać produkcję bazową. Jeśli niedopasowanie produkcji pochodnej jest związane z dodatkowymi węzłami i atrybutami dodanymi w produkcji pochodnej, to blokujemy wykonanie produkcji bazowej, jeśli natomiast z uściśleniem etykietowania węzłów występujących także w produkcji bazowej, to nie blokujemy jej wykonania.

Parametr **Override** równy 2 oznacza, że dopasowanie produkcji pochodnej blokuje możliwość wyboru produkcji bazowej, czyli jeśli została dopasowana pochodna, to gracz może ją wykonać, ale nie może wykonać bazowej. Jeżeli jednak produkcja pochodna nie została dopasowana a bazowa pasuje, to gracz zawsze może wykonać bazową. Taka sytuacja zazwyczaj ma miejsce, gdy gra ma nas karać za wystąpienie niekorzystnego dla nas modyfikatora (np. zakup z prowizją dla pośrednika) lub jeżeli produkcja pochodna daje nam dodatkowe możliwości bezwzględnie korzystne dla gracza (np. zakup z automatycznie naliczonym bonusem).

Mamy więc kilka możliwych sytuacji.

Produkcja szczegółowa vs inna produkcja szczegółowa, obie pasują. Jeżeli dwie różne produkcje szczegółowe otwierają dwie linie fabularne, to wyboru pomiędzy nimi powinien dokonać gracz. Np. mając owcę, siarkę i miecz może zdecydować się otruć smoka lub rozpocząć z nim walkę.

Produkcja szczegółowa vs jej pierwotna produkcja generyczna, obie pasują. W tym wypadku możemy dać graczowi wybór (parametr `Override` ustawiony na 0) lub nadać produkcji szczegółowej wyższy priorytet niż produkcji ogólnej (parametr `Override` równy 1 lub 2). To drugie wyjście może być związane z logiką fabularną: jeśli do świata pasuje produkcja ucieczki z więzienia, to nie będziemy z niego wychodzić spacerkiem. Jeśli zmiana lokacji bohatera chorego łączy się z utratą punktów HP, to nie dajemy mu możliwości przechodzenia pomiędzy lokacjami bez konsekwencji. Możemy też ograniczyć wybór gracza nie dlatego, że zaburzałby logikę świata, ale dlatego, że jest oczywisty i nie ma sensu zanudzać gracza pytaniami. Wyobraźmy sobie np. ciąg produkcji: `Walka z przeciwnikiem` → `Walka z przeciwnikiem z użyciem broni` → `Walka z przeciwnikiem z użyciem miecza`. Załóżmy, że broń się nie zużywa w trakcie walki a jej użycie zmniejsza nasze straty punktów HP. Załóżmy też, że użycie miecza dodatkowo zwiększa nasze punkty doświadczenia bitewnego. Czy jest sens pytać gracza, którą z trzech opcji chce wybrać: najkorzystniejszą, korzystną, czy najmniej korzystną? Ostateczną decyzję na temat szczegółowości wyborów gracza podejmuje oczywiście projektant, ustawiając właściwą wartość parametru `Override`.

Produkcja generyczna pasuje, produkcja pochodna nie pasuje. W takiej sytuacji mamy do wyboru: zablokować możliwość wybrania przez gracza produkcji generycznej ustawiając parametr `Override` produkcji pochodnej na 1 (ma to sens w przypadku niepasującej produkcji ucieczki z więzienia i pasującej zmiany lokacji) albo pozwolić mu ją wykonać ustawiając parametr `Override` produkcji pochodnej na 2 (ma to z kolei sens przy produkcji zmiany lokacji chorego bohatera, kiedy brak choroby nie powinien blokować wykonania zwykłej produkcji, wręcz przeciwnie).

Przyjrzyjmy się teraz sytuacji, gdy wykorzystanie produkcji generycznej w przypadku niemożności wykonania produkcji szczegółowej niweczy sens definiowania tej ostatniej. Niech bohater znajdzie się w więzieniu i zapragnie z niego uciec. Zdefiniowaliśmy do tego specjalną produkcję „Ucieczka z więzienia”, ustawiając jej parametr `Override` na 1 (por. kod 19).

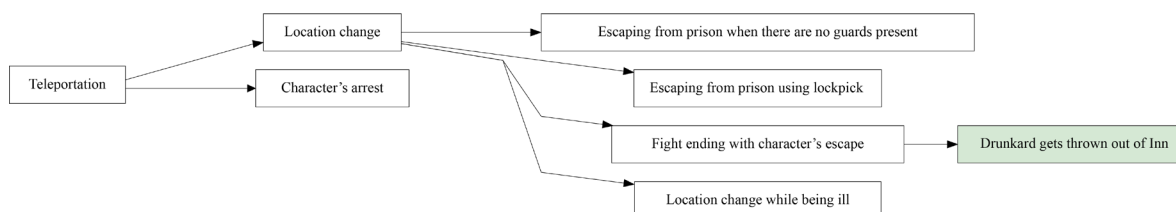
Zauważmy, że w przypadku gdybyśmy nie zablokowali wykonania produkcji generycznej „Zmiana lokacji”, to tworzenie produkcji „Ucieczka z więzienia” traci sens. Zawsze bowiem, gdy aresztowany bohater miałby ochotę odzyskać wolność, wybrałby jedną z dwóch produkcji: albo ucieka z więzienia zużywając wytrych, albo po prostu wychodzi.

Kod 19. Produkcja pochodna „Ucieczka z więzienia”. Na żółtym tle parametr Override oraz te fragmenty produkcji, które zostały dodane do produkcji generycznej „Zmiana lokacji” czyniąc nową istotnie od niej różną.

```
{
  "Title": "Escape from prison / Ucieczka z więzienia",
  "TitleGeneric": "Location change / Zmiana lokacji",
  "Description": "«Escapee» ucieka z więzienia do «Somewhere» wykorzystując wytrych.",
  "Override": 1,
  "LSide": {
    "Locations": [
      {
        "Name": "Prison",
        "Characters": [
          {
            "Id": "Escapee",
            "IsObject": true,
            "Items": [
              {
                "Name": "Lockpick"
              }
            ]
          }
        ]
      }
    ],
    "Connections": [
      { "Destination": "Somewhere" }
    ]
  },
  {
    "Id": "Somewhere"
  }
],
  "RSide": { },
  "Instructions": [
    { "Op": "move", "Nodes": "Escapee", "To": "Somewhere/Characters" },
    { "Op": "delete", "Nodes": "Lockpick" }
  ]
}
```

Przesłanianie produkcji

Przeanalizujemy wszystkie opisane powyżej warianty na autentycznym przykładzie misji zaprojektowanej przez studentów i produkcji pochodnych dla produkcji „Teleportacja” (por. rys. 30).



Rys. 30. Fragment drzewa hierarchii produkcji zawierający produkcje o różnych stylach dziedziczenia, produkcje pochodne produkcji generycznej „Teleportacja”.

Ta najbardziej bazowa produkcja wynika bezpośrednio z mechaniki gry i oznacza bezwarunkowe przeniesienie węzła z warstwy „Characters” zagnieżdżonego w jednym węźle warstwy „Locations” do drugiego. Pochodne po niej są dwie produkcje: „Location change” i „Character’s arrest”. Obie mają klucz **Override** ustawiony na wartość 1. W żadnej nie mamy uściślenia etykietowania, więc wyjątek zastosowania produkcji bazowej mimo **Override** 1 nie zachodzi. Oznacza to, że produkcja teleportacji w stanie czystym nigdy się nie wykona. W przypadku zaś, kiedy bohater spełni warunki obu produkcji pochodnych, będzie miał wybór, czy kogoś aresztować, czy zmienić lokację. Zmiana lokacji z kolei ma cztery produkcje pochodne, w dwóch pierwszych **Override** przyjmuje wartość 1, co oznacza blokowanie produkcji bazowej, czyli postać, która znalazła się w więzieniu, musi doprowadzić do dopasowania którejs z tych dwóch produkcji, żeby je opuścić. Trzecia produkcja pochodna „Fight ending with character’s escape” ma wartość **Override** równą 0, co oznacza, że gracz będzie miał wybór, a co za tym idzie, że produkcja „Location change” będzie mogła być wykonywana w grze na równych prawach, co jej pochodne. Ostatnia produkcja, „Location change while being ill” ma z kolei klucz **Override** ustawiony na 2, co oznacza, że w wypadku, kiedy się dopasują obie produkcje, to wykona się właśnie ona, ale w przypadku dopasowania tylko bazowej „Location change” nie zostanie ona zablokowana i może zostać wykonana.

Ostatnią produkcją jest „Drunkard gets thrown out of Inn”, która jest produkcją szczegółową, węzłową dla misji „Dragon Story”. Ma ona parametr **Override** równy 2, co oznacza, że gracz wykona właśnie ją, jeżeli będzie chciał skonfrontować się z pijakiem, ale brak któregoś dodatkowego komponentu w stosunku do produkcji bazowej nie zablokuje zwykłej karczemnej bijatyki.

Mechanizm przesłaniania produkcji bazuje na drzewie hierarchii produkcji, którego tworzenie omówione jest w rozdziale 5.1, podrozdział *Budowa drzewa hierarchii produkcji*.

Pierwsza produkcja pochodna to zmiana lokacji. Sprawdzamy jej dopasowania. Jeżeli nie ma żadnych, to nie sprawdzamy już pochodnych, a jeżeli są, to sprawdzamy ich dopasowania.

Pierwsza produkcja pochodna zmiany lokacji, „Escaping the prison using lockpick”, ma `Override` równe 1. Szukamy jej dopasowań.

Wariant pierwszy: znaleźliśmy dopasowanie dla produkcji pochodnej (np. w wariancie „Prison”, „Main_hero”, „Road”, „Lockpick”). Blokujemy te warianty dopasowania produkcji pierwotnej, które są podzbiorem dopasowań produkcji pochodnej („Prison”, „Main_hero”, „Road”). Poinformujemy gracza o możliwości wykonania produkcji pochodnej i jeśli ją wybierze, wykonujemy ją.

Wariant drugi: nie znaleźliśmy dopasowania produkcji pochodnej. Pamiętajmy, że `Override` z wartością 1 oznacza, że także w przypadku niedopasowania blokujemy produkcję pierwotną. Blokujemy te warianty dopasowania produkcji pierwotnej, które są podzbiorem dopasowań produkcji pochodnej. Jeżeli w zbiorze dostępnych wariantów produkcji „Location change” jest zestaw („Prison”, „Main_hero”, „Road”), a niedopasowanie produkcji pochodnej wynika z braku węzła „Lockpick”, to blokujemy produkcję bazową. Jeśli w zbiorze węzłów dopasowanych znalazł się inny zestaw (np. „Inn”, „Main_hero”, „Road”) to nie blokujemy jej. Czyli bohater, który jest w więzieniu i nie ma wytrychu, pozostaje w nim bez możliwości wyjścia a bohater, który jest w karczmie, może wyjść z niej w każdej chwili i nie zostanie zablokowany przez niedopasowaną produkcję „Escaping the prison using lockpick”.

Analogicznie postępujemy w przypadku drugiej pochodnej produkcji zmiany lokacji, która ma `Override` równe 1, „Escaping the prison when there are no guards present”.

Ostatnią produkcją pochodną dla „Location change” jest „Location change while being ill”. Sprawdzamy, czy się dopasuje, jeśli tak, to usuwamy ze zbioru wariantów pierwotnej produkcji te, które zawierają się w zbiorze wariantów pochodnej. Jeśli nie, to nie zmieniamy niczego.

Tak zbudowaną listę dostępnych produkcji udostępniamy graczowi w całości lub we fragmentach, jeśli w implementacji wprowadzono pojęcie kontekstu pozycji gracza na ekranie, czyli wyświetlania graczowi tylko produkcji wykorzystujących węzły wyrenderowane na ekranie w odległości nie większej niż zadana.

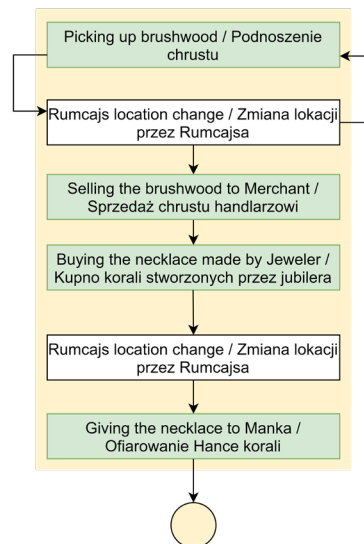
Świat otwarty vs świat zamknięty

Prześledźmy różnice między grą w świecie zamkniętym i otwartym na przykładzie prostej misji „Zakup naszyjnika” w świecie opartym na bajce *Rumcajs* z tytułowym rozbójnikiem jako głównym bohaterem.

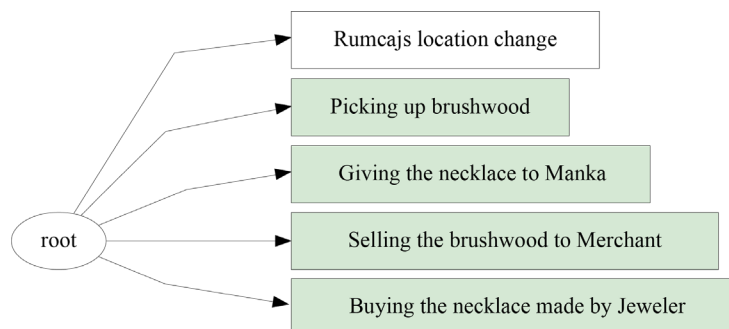
W pierwszym wariantcie (rys. 31) każda produkcja z wyjątkiem zmiany lokacji jest produkcją całkowicie szczegółową. Produkcje nie wywodzą się jedne z drugich, drzewo hierarchii (rys. 32) jest bardzo płaskie. Gracz może wykonać tylko działania zaprojektowane w podstawowej wersji misji a każda produkcja była tworzona specjalnie na potrzeby historii. Gracz nie może zrobić nic poza odgadnięciem i odtworzeniem pomysłu projektanta: znajdź chrust, sprzedaj kupcowi zdobywając monety, kup naszyjnik.

W drugim z kolei wariantcie (rys. 33) nie stworzono od podstaw ani jednej nowej produkcji, wykorzystano produkcje generyczne z gry „Royal Envoy”. Hierarchia produkcji jest bardziej rozbudowana (rys. 34) i żaden węzeł w lewej stronie produkcji nie jest etykietowany. Produkcji jest więcej, ale żadna nie została stworzona wyłącznie na potrzeby tej misji.

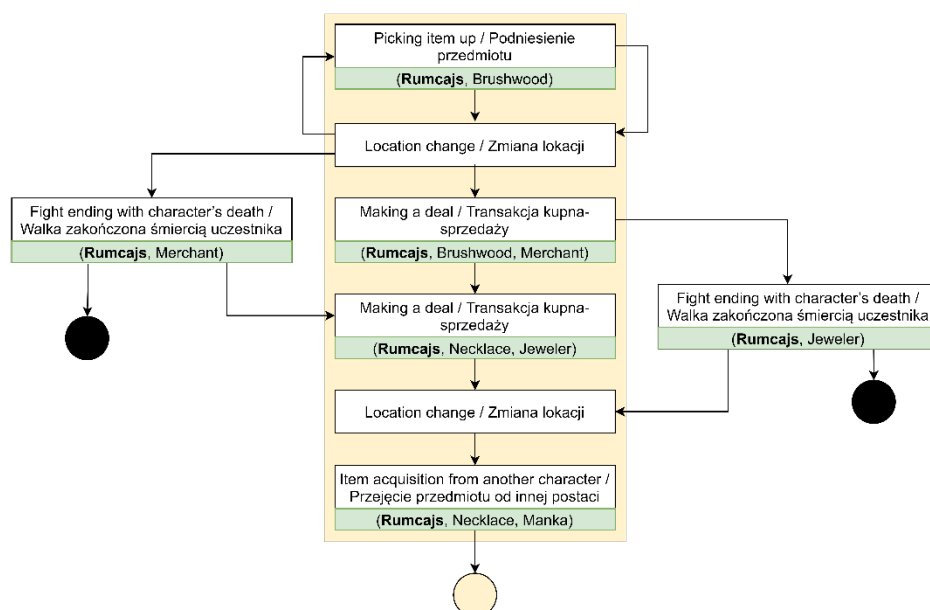
Zobaczmy, jak bardzo poszerzyły się możliwości gracza w stosunku do pierwszego wariantu. Rumcajs może pozbierać chrust i sprzedać kupcowi a za tak uzyskane pieniądze kupić naszyjnik od jubilera (czyli odtworzyć wariant pierwszy), może go sprzedać bezpośrednio jubilerowi, gdyby ten potrzebował opału, może też (w końcu jest rozbójnikiem) obrabować kupca i zdobyte w ten sposób łupy sprzedać jubilerowi albo obrabować jubilera i tak zdobyty naszyjnik zanieść Hance. Może też zanieść Hance drewno i zaproponować, żeby poszła z nim do miasta, sprzedała i kupiła sobie coś ładnego. Możemy też rozegrać tę grę z pozycji Hanki, zebrać drewno i samodzielnie pójść do miasta nie oglądając się na męża.



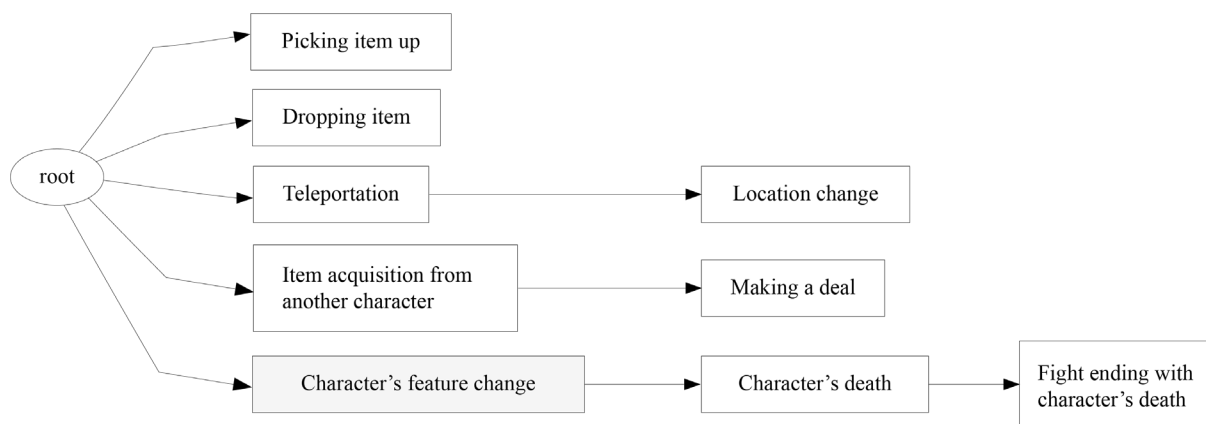
Rys. 31. Misja zdobycia naszyjnika oparta o produkcję szczegółową.



Rys. 32. Drzewo hierarchii produkcji dla misji opartej o produkcję szczegółową.



Rys. 33. Misja zdobycia naszyjnika oparta o produkcję generyczną.



Rys. 34. Drzewo hierarchii produkcji dla misji opartej o produkcje szczegółowe.

Odtwarzanie i symulowanie rozgrywki

W modelu grafowym każde działanie postaci wyrażone jest poprzez nazwę produkcji i listę węzłów dopasowanych do lewej strony produkcji. Dzięki temu możemy w skróconej formie zapisywać komplet informacji potrzebny do odtworzenia tej akcji. W implementacji referencyjnej logi gry inicjowane są zapisem początkowego stanu świata i zbioru produkcji (dzięki temu możemy odtworzyć rozgrywkę z danego logu niezależnie od ewentualnych zmian plików źródłowych) a następnie po każdym ruchu dopisywane są informacje o wykonanej produkcji i sygnatura czasowa. W momencie składania niniejszej pracy powstaje praca magisterska, która pozwoli wygenerować na podstawie logu animację pokazującą rozgrywkę albo w czasie rzeczywistym (z uwzględnieniem sygnatur czasowych) albo w czasie przyspieszonym. W przyszłości podczas odtwarzania rozgrywki możemy do takiej wizualizacji dodać widok śledzenia ścieżki fabularnej.

Można też śledzić rozgrywkę za pomocą interfejsu czysto tekstowego. W referencyjnej implementacji jest to dość prosty interfejs bazujący na szablonach komunikatów dla gracza opartych o elementy produkcji (i umieszczający w nich odpowiednio szczegółowe informacje dotyczące węzłów dopasowanego wariantu), ale planowane jest rozbudowanie tego modułu o szerszy kontekstowy opis sceny.

W przypadku zapisu rozgrywki do analizy działań użytkownika i wnioskowania możemy dodać do logu różne informacje, np.:

- informacje potrzebne do działania algorytmu *Bayesian Knowledge Tracking* (w ramach tworzonej implementacji gry poważnej, por. rozdział 6.3)

- informacje o pozycji graficznych reprezentacji obiektów ze świata gry na ekranie (pozwoli to na bardziej dokładne symulowanie wyglądu rozgrywki ale też na analizę zachowań użytkownika)
- informacje na temat fiksacji wzroku użytkownika na poszczególnych elementach interfejsu w trakcie rozgrywki (pozwoli to na prześledzenie związków między percepcją użytkownika a decyzjami fabularnymi). Badania okulograficzne (ang. *eye tracking*) wykorzystywane są coraz częściej do analizy działań gracza (por. Grabska-Gradzińska, 2016, Grabska-Gradzińska, Argasiński, 2017), połączenie danych okulograficznych z informacjami z modelu grafowego pozwoli głębiej wejść w strukturę rozgrywki w procesie analizy percepcji.

System StoryGraph nie jest przeznaczony do automatycznego tworzenia fabuł lecz do wspomagania twórcy w procesie projektowania, należy zwrócić jednak uwagę, że jest możliwe automatyczne tworzenie linii fabularnych jako ciągu wyborów dokonanych automatycznie. Tak zapisane rozgrywki także można odtwarzać lub analizować na podstawie wygenerowanego zapisu, analogicznie jak w przypadku rozgrywek wykonanych przez żywego gracza.

6.2 Proces projektowy

Sformalizowane modele projektowe przydają się szczególnie wtedy, gdy nad danym dziełem pracuje wiele osób. W przypadku gier naturalnym sposobem rozdzielenia pracy nad fabułą jest podzielenie jej na krótkie historie, zwane misjami (ang. *quests*). Odbywa się to oczywiście po wstępnym nakreśleniu ram fabularnych i wyznaczeniu ostatecznego celu gry.

W procesie pracy nad grą biorą udział trzy grupy uczestników: twórcy świata, twórcy fabuły oraz twórcy interfejsu (projektanci UX i graficy). Pierwszym krokiem prac jest stworzenie obiektów świata gry i możliwych typów interakcji. Ich odwzorowaniem w grafie są węzły GWSG i produkcje generyczne. Tym zajmują się twórcy świata, którzy określają także warunki początkowe i końcowe gry, tzn. produkcję zawiązującą fabułę i produkcję finałową. W schemacie te elementy, oznaczone jako: PG, P_{in}, P_{out} i piktogramy obiektów oznaczone są kolorem fioletowym (zob. punkt 1 na rys. 35).

W określonych przez twórców świata ramach pracują projektanci historii, tworząc poszczególne misje. Misja jest fragmentem fabuły gry pomyślanym jako spójna całość. Wejście w misję, czyli wykonanie produkcji inicjującej, może wymagać spełnienia jakichś warunków (np. będących efektem innej misji, poprzedzającej). O wykonaniu misji mówimy, gdy zostanie

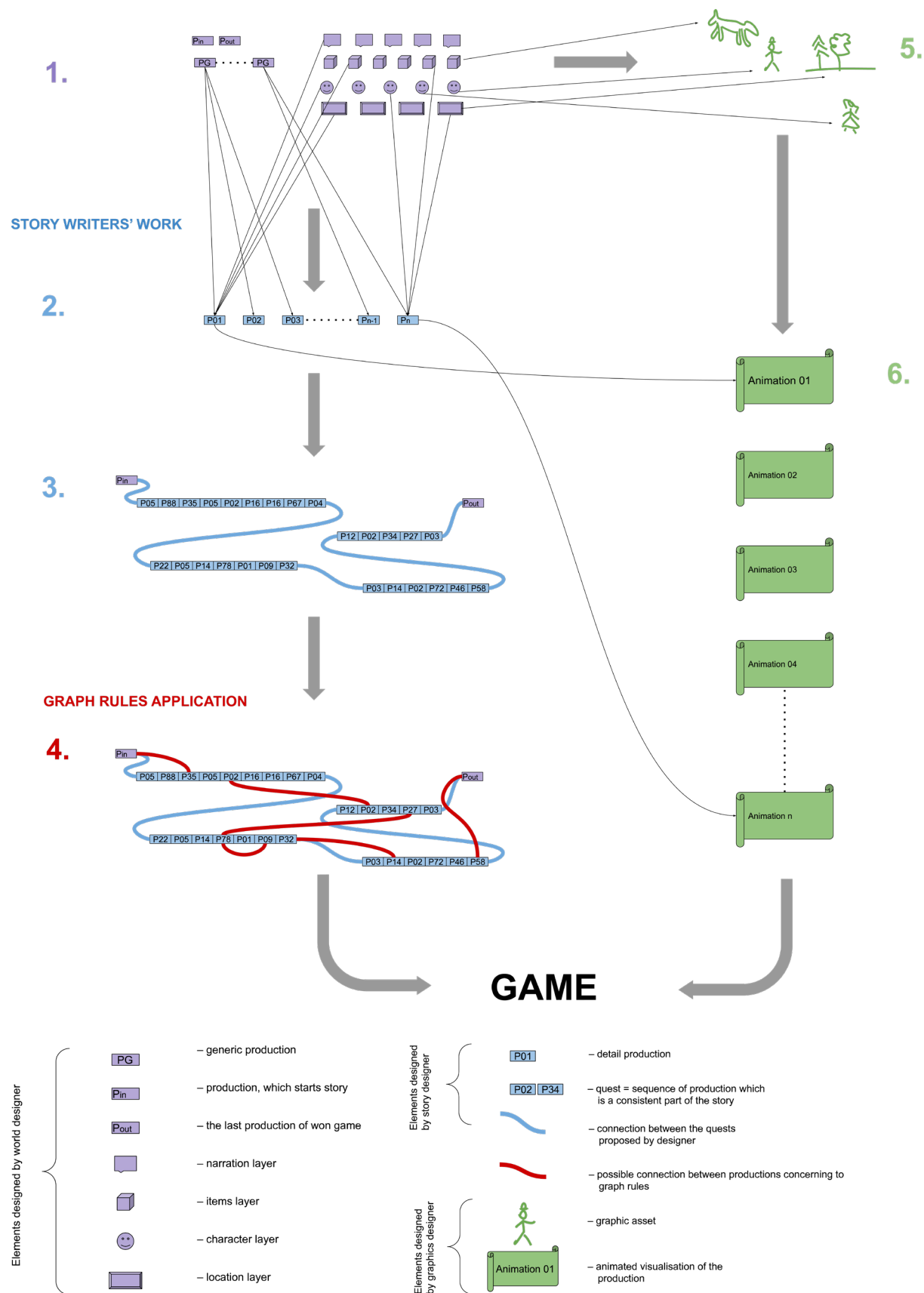
osiągnięta produkcja końcowa misji. Rozgrywka w ramach misji może być liniowa, ale może się też rozgałęziać. Projektanci misji na podstawie produkcji generycznych tworzą węzłowe produkcje szczegółowe, dobierając do nich odpowiednie obiekty świata gry i przypisując im atrybuty wymagane przez fabułę (zob. punkt 2 na rys. 35). Projektanci tak przygotowują produkcje szczegółowe, aby wraz z produkcjami generycznymi tworzyły ciąg przyczynowo-skutkowy przynajmniej dla jednej wersji misji (zob. punkt 3 na rys. 35).

Pomocą w tworzeniu misji jest diagram projektowy, czyli schematyczny wykres rozgrywki odwzorowujący najważniejsze warianty przejścia misji. Oprócz samej sekwencji działań w misji zawiera on informacje o zazębianiu się misji, czyli efektach jednej misji, które stają się warunkami inicjalnymi drugiej.

Logika modelu grafowego powoduje, że powstałe produkcje szczegółowe mogą być wykonywane na grafie także w innej kolejności niż ta zaproponowana przez twórcę danej misji, jeśli tylko uda się je w takiej kolejności dopasować. Umożliwia to graczowi przejście gry w inny sposób niż pierwotnie zaplanowany i pozwala osiągnąć pożądaną różnorodność fabularną rozgrywek. Każda wykonywalna sekwencja produkcji odpowiada jednej potencjalnej rozgrywce (zob. punkt 4 na rys. 35). Zadaniem twórców produkcji jest takie ich napisanie, aby gra w żadnej konfiguracji nie stała się trywialna lub nie doprowadziła gracza do pętli bez wyjścia.

WORLD DESIGNERS' WORK

GRAPHIC DESIGNERS' WORK



Rys. 35. Schemat współpracy projektantów gry w systemie.

Równolegle do projektantów historii pracują graficy. Na podstawie obiektów świata gry przygotowanych przez twórców świata przygotowują materiały graficzne (ang. *assets*) dla każdej postaci, lokacji i przedmiotu (zob. punkt 5 na rys. 35). Z kolei po przygotowaniu produkcji szczegółowych powstają – w oparciu o materiały graficzne – animacje odpowiadające każdej produkcji szczegółowej (zob. punkt 6 na rys. 35). Animacje zaprojektowane są tak, żeby mogły płynnie łączyć się w ciągi. Pozwoli to na automatyczne wygenerowanie filmu odpowiadającego rozgrywce.

Projektowanie pojedynczej misji

Przeanalizujmy przykładową misję „Dragon Story”. Przedstawiony poniżej diagram projektowy jest przykładem języka wizualnego wypracowanego na potrzeby współpracy studenckich grup projektowych. Szczegóły rozwiązań wizualnych są oczywiście niezależne od modelu grafowego, służą skutecznemu porozumiewaniu się grup projektowych między sobą, ale generalna zasada budowy takiego diagramu pozostaje w ścisłym związku z modelem: każda ścieżka diagramu to potencjalna rozgrywka, czyli ciąg produkcji ułożony poprzez dopasowanie zmian w świecie wynikających z poprzedniej produkcji z lewą stroną następnej.

Przykład: produkcja końcowa misji to produkcja podniesienia smoczego jaja z jaskini smoka przez głównego bohatera. Próba wykonania jej bez wykonania produkcji poprzedzających w misji zakończy się śmiercią bohatera ze względu na obecność smoka w jaskini. Czyli aby zdobyć smocze jajo bohater musi pokonać smoka. Nie jest w stanie wygrać walki ze smakiem z marszu, więc przygotowanie sobie przewagi nad smakiem jest w zasadzie przedmiotem tej misji.

Misja byłaby trywialna, gdyby początkowe atrybuty i stan posiadania bohatera umożliwiała mu wykonanie produkcji wieńczącej misję w dowolnym momencie gry bez przygotowania innego niż przejście do odpowiedniej lokacji.

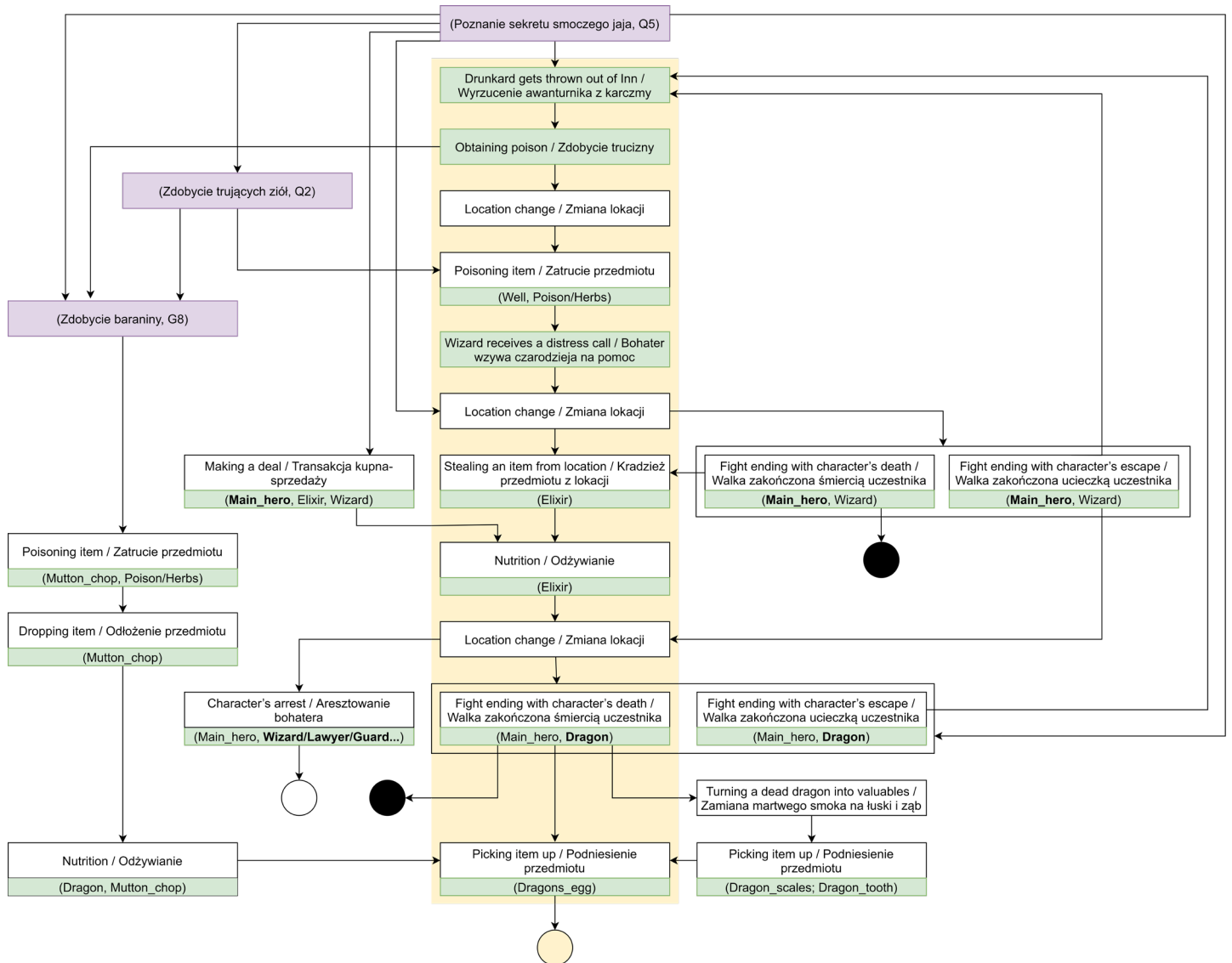
Najczęściej wyróżniamy jeszcze produkcję inicjującą, która umożliwi albo ułatwi wejście w misję. Albo jest ona warunkiem wejścia w misję i wtedy jest pierwszą produkcją misji albo nie jest niezbędna do wykonania misji, ale skłania gracza do podjęcia decyzji umożliwiających wejście w misję.

W przykładowej misji takim wydarzeniem inicjującym będzie zdobycie przez gracza informacji o tym, że w jaskini smok wysiaduje cenne smocze jajo. Nie jest to warunek konieczny wykonania misji – gracz znający popkulturę, np. opowiadanie Andrzeja Sapkowskiego *Granica możliwości*, może wejść w misję na podstawie swojej wiedzy ogólnej o potencjalnie cennej wartości smoczej jaskini bez dodatkowych zachęt w samej fabule gry.

Ciąg produkcji poprzedzający produkcję końcową może być jednoznacznie określony (wtedy misja ma strukturę liniową) albo może być wariantywny w zależności od stanu posiadania i własności bohatera w momencie rozpoczęcia misji oraz temperamentu gracza. Wtedy jednak jeden wariant misji traktujemy jako ten podstawowy. Zazwyczaj projektowanie rozpoczynamy od przygotowania wariantu podstawowego a potem rozszerzamy projekt o warianty poboczne.

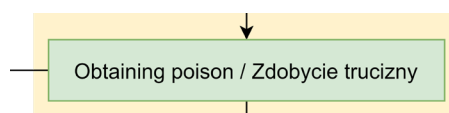
Dla ułatwienia procesu projektowania i późniejszej analizy gry tworzymy schemat misji w postaci diagramu projektowego. Nie jest to diagram sterujący w tym sensie, że nie zawiera zamkniętego zbioru możliwości przejścia misji, ale zawiera wszystkie ścieżki przejścia misji, które projektant wziął pod uwagę i uznał za istotne.

Rysunek 36 przedstawia schemat misji „Dragon Story”. Na żółtym tle widzimy podstawowy wariant misji, ciąg produkcji zakończonych produkcją kończącą „Picking item up (Dragon’s egg)”. W fioletowych prostokątach zaznaczone są poprzedzające misję działania inicjujące, które jednak nie są elementem samej misji, ale ułatwiają wejście do niej. W ich opisie dodana jest także informacja o numerze misji, w której takie działanie inicjujące mogło się wydarzyć. Fioletowe prostokąty mają charakter jedynie informacyjny dla projektanta. Na zielono zaznaczone są produkcje szczegółowe, zaprojektowane specjalnie na potrzeby tej misji, a na białą generyczne. Zielony pasek pod produkcją generyczną wskazuje sposób jej uszczegółowienia (por. rysunki 37–39).



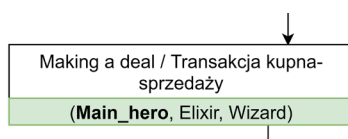
Rys. 36. Schemat misji „Dragon Story”.

Patrząc na diagram widzimy, że w tej misji mamy cztery produkcje szczegółowe: „Drunkard gets thrown out of Inn”, „Obtaining poison”, „Wizard receives a distress call from Main hero” i „Turning a dead dragon into valuables”.



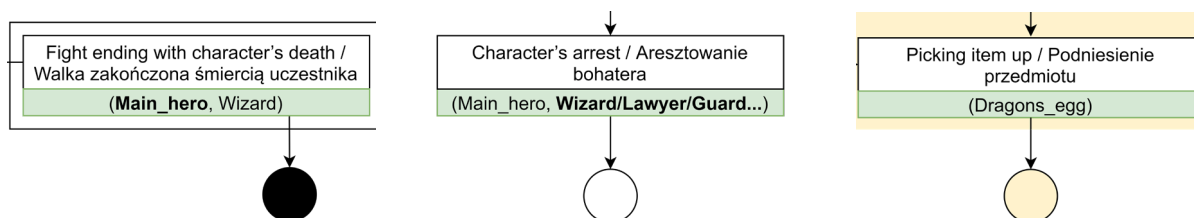
Rys. 37. Fragment schematu. Przykład produkcji szczegółowej.

Pozostałe produkcje wykorzystane w misji to produkcje generyczne, których nie musimy dodatkowo definiować: „Location change”, „Poisoning item”, „Stealing an item from location”, „Fight ending with character’s death”, „Fight ending with character’s escape”, „Making a deal”, „Picking item up”, „Dropping item”, „Nutrition”, „Character’s arrest”. Widzimy je jako białe prostokąty. W schemacie wskazujemy, jakie konkretnie elementy mają zastąpić ogólne etykiety w tych produkcjach, czyli wskazujemy, w jaki sposób mają zostać uszczegółowione. Zaznaczone jest to w zielonym prostokącie poniżej białego. Jeżeli w produkcji bierze udział dwóch lub więcej bohaterów, to pogrubieniem wskazujemy, który jest inicjatorem działania.



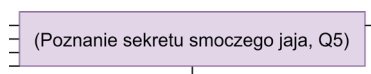
Rys. 38. Fragment schematu. Przykład produkcji generycznej i sposobu jej uszczegółowienia. Pogrubiony podmiot akcji.

Kółka wskazują moment zakończenia misji. Czarne kółka wskazują wariant, w którym bohater ginie a białe wariant, w którym bohater nie ginie, ale nie może kontynuować misji. Żółte kółko oznacza sukces misji.



Rys. 39. Fragmenty schematu. Zakończenia misji: a) porażka: śmierć bohatera, b) porażka: niemożność dokończenia misji, c) sukces.

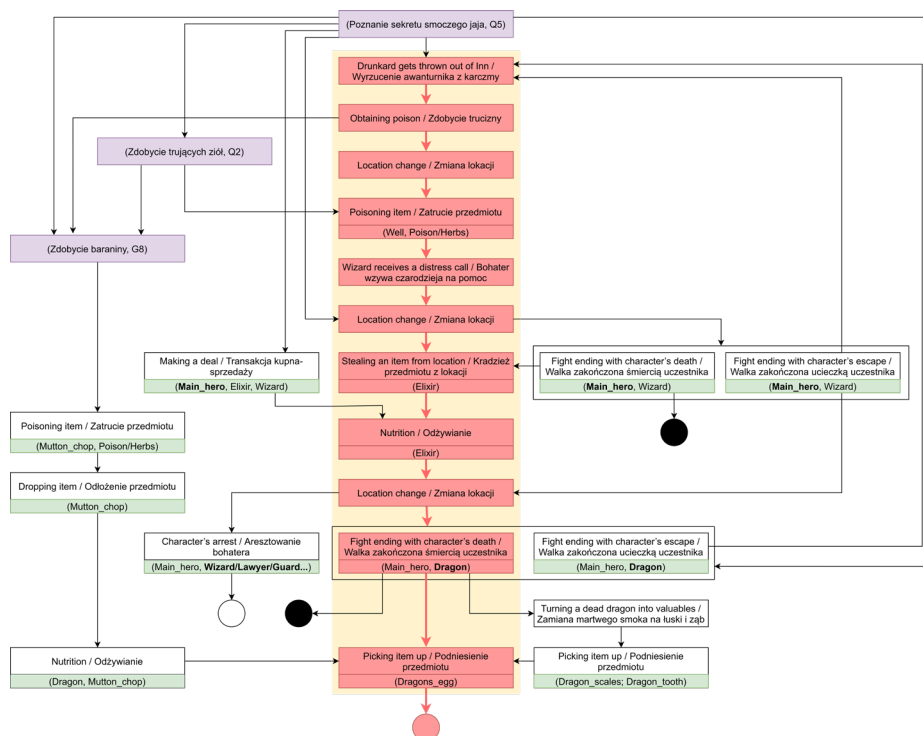
Przed wejściem w misję gracz może wykonać działania poprzedzające, które są zachętą do wejścia w misję lub dostarczą graczowi możliwości wejścia w środek misji, z pominięciem kilku produkcji początkowych.



Rys. 40. Fragment schematu. Działanie poprzedzające, czyli zdobycie wiedzy zachęcającej do wejścia w misję. Jest możliwe do wykonania w misji nr 5, czyli Q5.

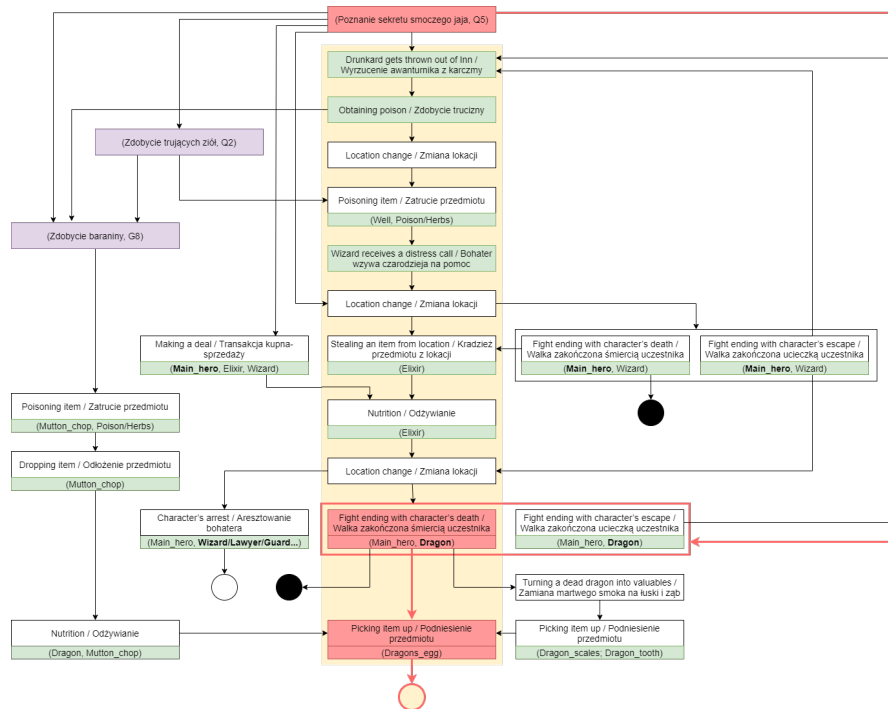
Powyższy schemat nie jest *sensu stricto* diagramem sterującym, ponieważ zawiera tylko niektóre warianty fabuły i nie jest wymagany do sterowania postaciami gracza. Diagramy sterujące będą za to podstawą do działań bohaterów niezależnych, ponieważ tam nie ma mowy o decyzyjności a jedynie o zbiorze dostępnych akcji i schemacie ich wykonywania. Gracz w każdym momencie gry może wykonać wszystkie dostępne (spełniające wszystkie ograniczenia) dla jego postaci w danej lokacji produkcje i podejmuje autonomiczne decyzje niezależne od sekwencji akcji będącej wizją projektanta. Projektant przewiduje tylko niektóre warianty rozwoju akcji, stara się jednak zaznaczyć na schemacie misji wszystkie te, które w istotny sposób wpływają na grywalność gry.

Przeanalizujmy warianty wyróżnione na rysunkach 41–45. Na czerwono zaznaczymy sekwencję produkcji odpowiadającą jednemu wariantowi.

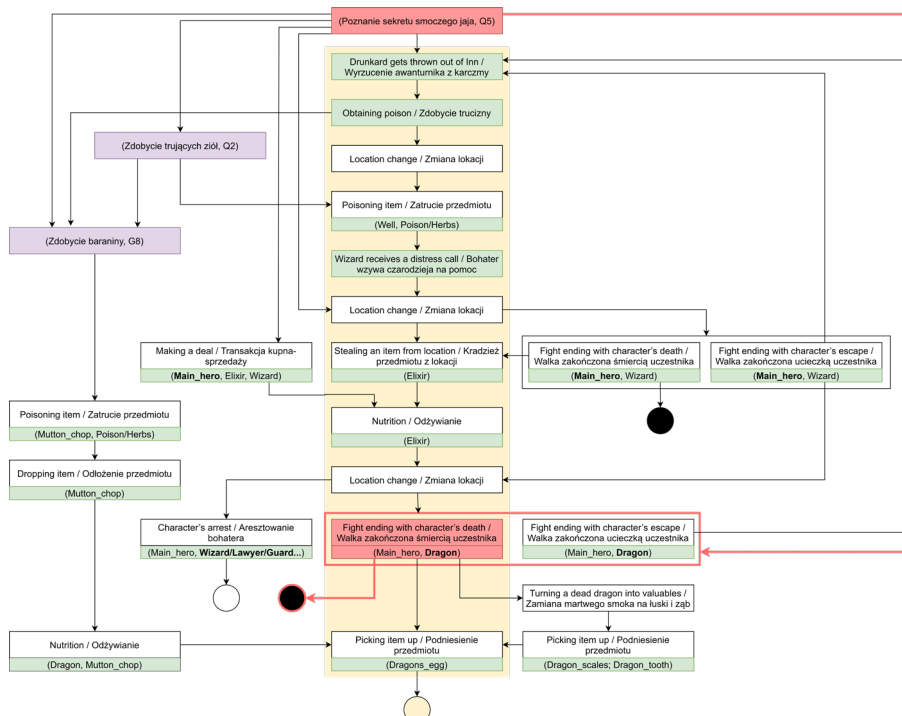


Rys. 41. Wariant podstawowy wykonania misji uwzględniony w schemacie.

Wariantem podstawowym jest ten na żółtym tle (por. rysunek bazowy 36 i rysunek 41 z zaznaczoną linią fabularną). Przedstawia on pewną spójną historię niemającą żadnych warunków początkowych (akcję poprzedzającą w postaci zdobycia informacji o smoczym jaju traktujemy jako zachętę a nie jako warunek konieczny) i kończącą się uzyskaniem korzyści przez gracza.

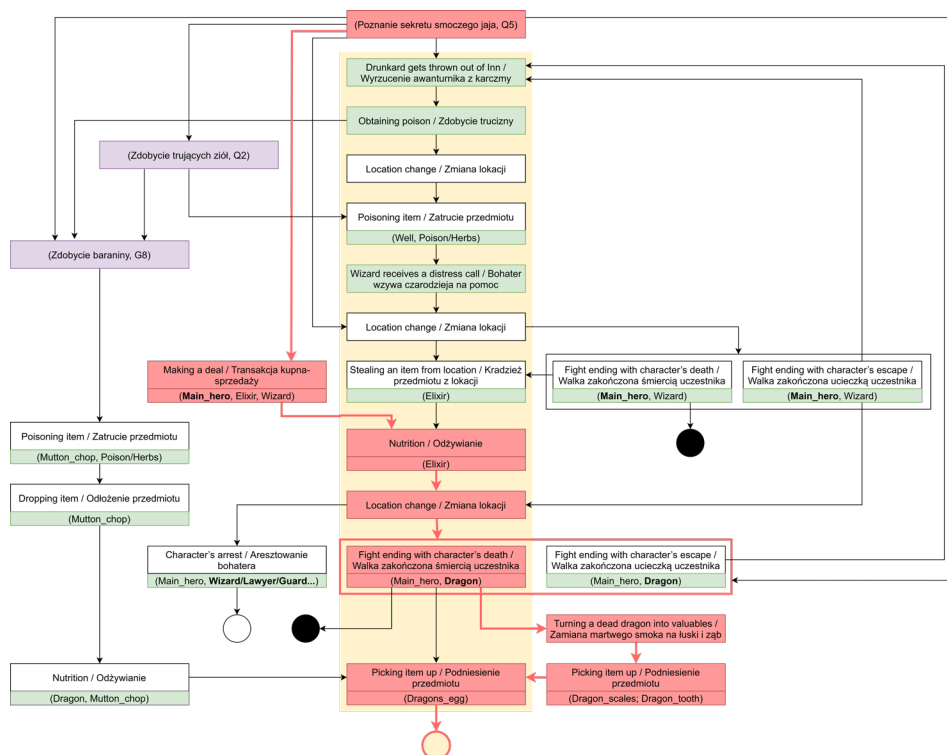


Rys. 42. Wariant trywialny wykonania misji zakończony sukcesem.



Rys. 43. Wariant trywialny wykonania misji zakończony klęską

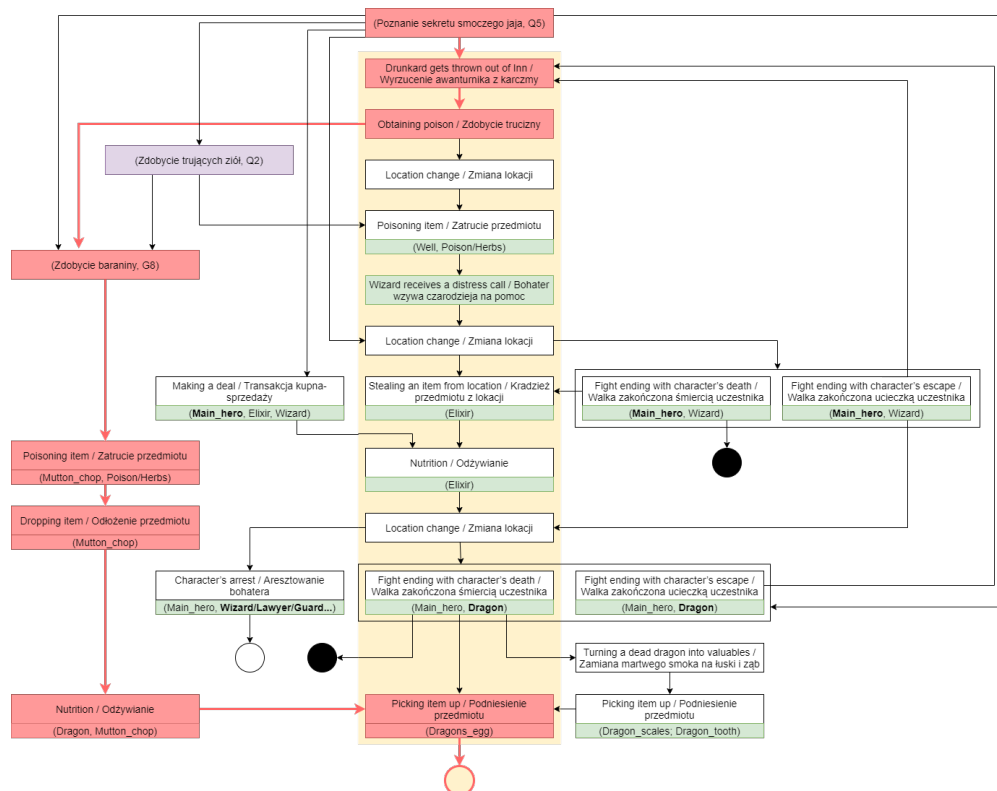
Wariantem trywialnym misji byłaby sekwencja **Walka ze smokiem** → **Podniesienie smoczego jaja** (rys. 42), dlatego w interesie projektanta gry istotne jest, aby taka sekwencja nieoprowadzona przygotowaniami kończyła się śmiercią bohatera (rys. 43).



Rys. 44. Wariant „legalistyczny” wykonania misji.

Wariantem rozgrywki zależnym od temperamentu gracza jest np. wariant, w którym gracz wchodzi w misję z pewnym zasobem gotówki i zamiast wykorzystać truciznę do zwabienia czarodzieja do wioski sprzedaje ją na targu, a dołożywszy uzyskane pieniądze do posiadanych środków legalnie kupi eliksir (rys. 44). Będzie uboższy, ale nie narazi się na ryzyko aresztowania po kradzieży. Przypuszczalnie taki gracz jednak nie tylko weźmie jajo smoka, ale też skrzętnie wybiera smocze zęby i smocze łuski, żeby też sprzedać je później na targu, dzięki czemu odzyska zainwestowane środki. Całą misję zaś przejdzie nie łamiąc prawa i nie narażając się na sankcje karne.

Wariantem rozgrywki świadczącym o dobrym osadzeniu gracza w popkulturze będzie natomiast sekwencja, w której gracz pozyskawszy truciznę zużyje ją na zatrucie owcy, którą postawi smokowi i dzięki temu uniknie walki ze smakiem, wcielając w życie pomysł z polskiej legendy ludowej obecny także w narracjach gatunkowych *fantasy* (rys. 45).



Rys. 45. Wariant wykonania misji wykorzystujący znajomość legendy.

Podstawy decyzji gracza

Gracz samodzielnie podejmuje autonomiczne decyzje wybierając w każdym kroku dowolną z dostępnych akcji. Gracz dokonuje decyzji na podstawie przesłanek z trzech kategorii. Po pierwsze, wiedzy wynikającej z relacji między elementami snopka, w którym się znajduje. Wiedzę tę zdobywa naocznie w każdym ruchu gry, ponieważ elementy snopka przedstawione są wizualnie lub tekstowo na scenie gry. Po drugie, samego pojawienia się wśród możliwości produkcji szczegółowych zaprojektowanych na potrzeby danej misji, co jest sugestią możliwego rozwoju akcji. Po trzecie, na podstawie wiedzy wynikającej z atrybutów warstwy narracyjnej, do których ma dostęp. Poszerzeniu tej wiedzy służą produkcje typu: „Rozmowa”, „Przeczytanie ogłoszenia” itp. Informacje te zawarte są w dzienniczku postaci, do którego gracz może w każdej chwili sięgnąć. Po czwarte, co jest najtrudniejsze do uwzględnienia w procesie projektowania, gracz może podejmować decyzje na podstawie swojej wiedzy ogólnej, znajomości konwencji gier przygodowych i znajomości popkultury.

Przykład: gracz widzi w karczmie bohatera awanturującego się (atrybut: `isTroublemaker`) i właściciela karczmy. Konwencja gier przygodowych podpowiada, że przysługa uczyniona bohaterowi dysponującemu jakimiś zasobami może przynieść korzyść i gracz podejmuje decy-

zję o wyrzucenia pijaka z karczmy. Faktycznie, zyskuje wdzięczność karczmarza, która pojawia się jako węzeł z warstwy narracyjnej związany z głównym bohaterem. Następnie, niekoniecznie natychmiast, bohater wykorzystuje tę wdzięczność, pozyskując od karczmarza truciznę. Za każdym razem, kiedy spotka karczmarza, wśród dostępnych produkcji pojawi się bowiem produkcja „Czy chcesz poprosić karczmarza o truciznę?”, mająca w lewej stronie węzeł „Wdzięczność karczmarza”. Wartość atrybutu **Knowledge** węzła „Wdzięczność karczmarza” widoczna jest w dzienniczku postaci, co może skłonić bohatera do aktywnego poszukiwania lokacji, w której spotka się z karczmarzem, gdyby nie skorzystał z oferty natychmiast.

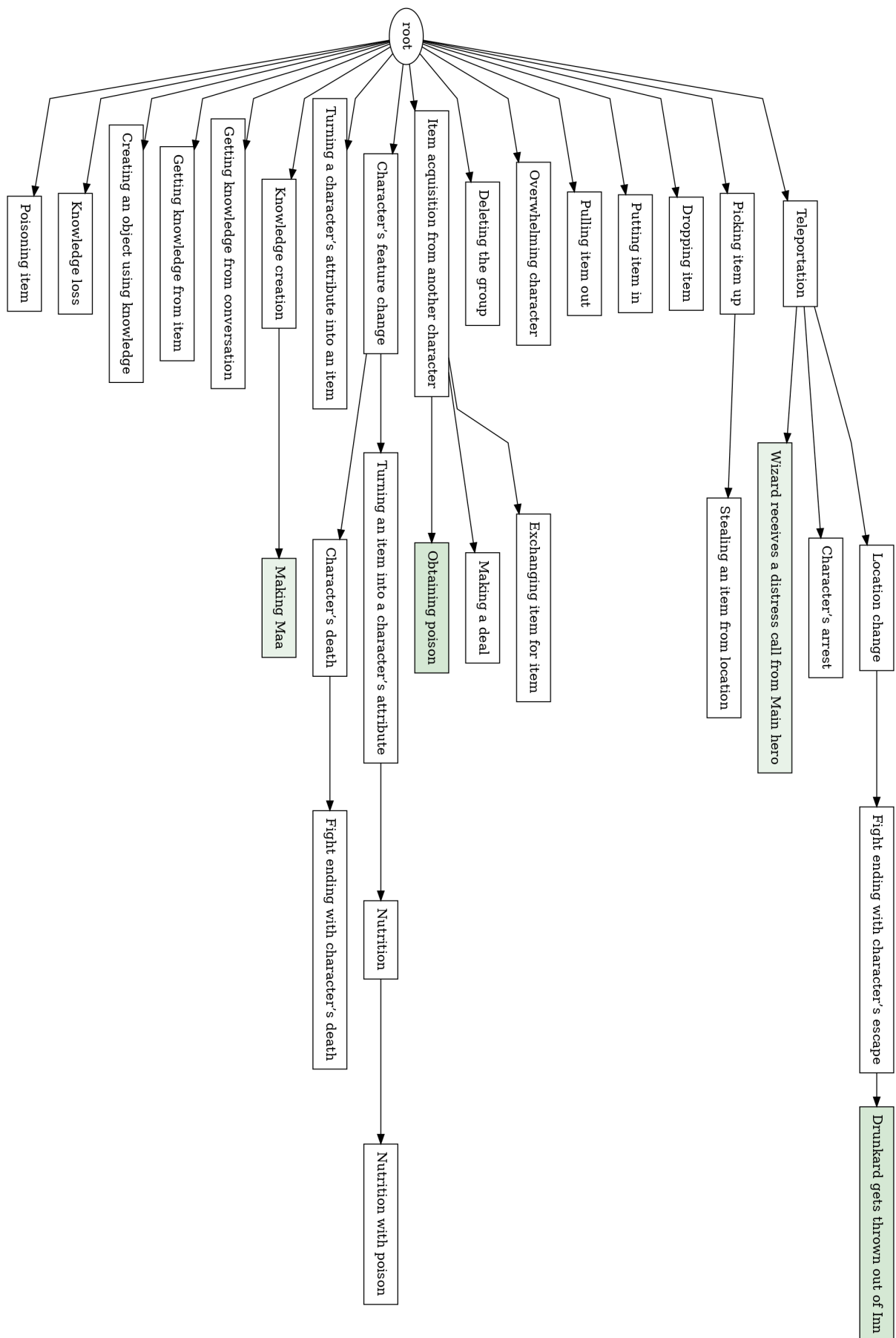
Decyzję, aby wykorzystać pozyskaną truciznę do zdobycia smoczego jaja, gracz może podjąć na podstawie swojej wiedzy ogólnej (smoki mają skarby w jaskiniach) lub na skutek wcześniejszej rozmowy z postacią, która przekaze mu plotki o smoczym jajku. Zwróćmy tutaj uwagę na różnicę między węzłem narracyjnym z informacją o sekrecie smoczego jaja, który ma na celu wyłącznie zachęcenie gracza do poszukiwania smoczej jaskini, a węzłem narracyjnym „Wdzięczność karczmarza”, który oprócz posiadania walorów informacyjnych stanowi wymóg formalny wykonania produkcji pozyskania trucizny.

Koncepcja, aby pozyskaną truciznę wykorzystać do zatrucia studni i wywabienia z chaty czarodzieja jest autorskim pomysłem projektanta i wymaga odpowiedzi dla gracza. Może taką odpowiedzią być anons na słupie ogłoszeniowym we wsi o treści: „Obywatele! Jeśli podejrzewacie zatrucie wody w studni, wezwijcie mnie na pomoc. Uzdatnię studnię za drobną opłatą 5 talarów oraz wikt i opierunek na trzy dni intensywniej pracy”. Być może na tym samym słupie powinno wisieć drugie ogłoszenie: „Obywatele! Elixiry przeciw smokom do nabycia w chacie czarodzieja, 100 talarów za pewność, że zgładzicie smoka w pierwszej walce”.

Koncepcję, aby wykorzystać truciznę do zatrucia owcy i podłożenia jej smokowi, gracz zna z legend. Dodatkową zachętą będzie płynąca ze snopka „Pastwisko” informacja o pasących się tam owcach, więc gracz, który albo przechodził przez pastwisko podczas poprzednich misji, albo specjalnie tam pójdzie zachęcony nazwą lokacji, może zdobyć baraninę i wykonać tradycyjny wariant zabicia smoka.

Pomysł, aby truciznę po prostu sprzedać i kupić eliksir wynika z wiedzy ogólnej bohatera i nie jest w żaden sposób sugerowany wprost, choć pomaga w podjęciu takiej decyzji informacja o cenie eliksiru.

Drzewo hierarchii produkcji misji „Dragon Story” przedstawione jest na rysunku 46.



Rys. 46. Drzewo hierarchii produkcji misji „Dragon Story”.

Zespołowe tworzenie fabuły „Royal Envoy”

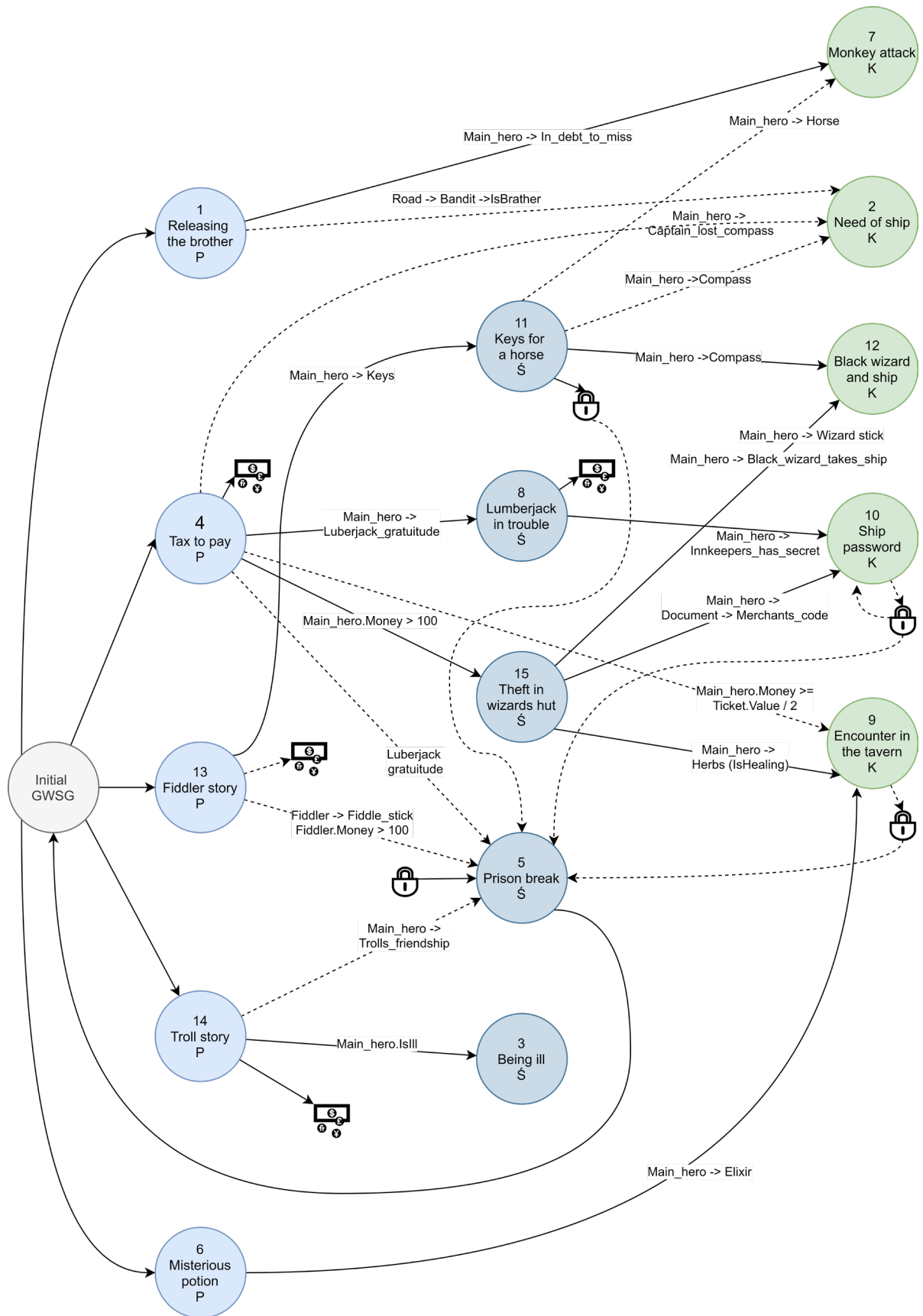
Przedstawiony w poprzednim rozdziale cykl prac projektowych został wykorzystany w praktyce podczas tworzenia fabuły gry przygodowej „Royal Envoy”, tworzonej przez studentów informatyki na zajęciach „Projektowanie wspomagane komputerem” w roku akademickim 2021–2022. Zespół projektowy liczył 42 osoby. Spośród nich wyłoniono zespół projektantów świata, zespół testerski i zespoły projektantów historii, które stworzyły 15 misji.

Zespoły zostały przypisane do jednej z trzech kategorii: misji początkowych, czyli możliwych do przeprowadzenia w świecie początkowym, misji środkowych, które wymagają zaistnienia któregoś ze stanów końcowych misji początkowej, aby było możliwe ich wykonanie oraz misji końcowych, które wymagają zaistnienia któregoś ze stanów końcowych misji środkowej i których efektem musiało być osiągnięcie celu gry. Podział ten miał uniemożliwić wystąpienie sytuacji, w której graczowi wystarczyłoby przejście jednej misji, aby zakończyć grę. Każda misja mogła się też zakończyć porażką, czyli śmiercią, cofnięciem rozwoju postaci (utrata środków, pieniędzy, wiedzy itp.) lub co najmniej utratą możliwości kontynuacji misji.

Wytyczne dotyczące ram fabularnych (lista produkcji generycznych, układ lokacji i lista dostępnych etykiet postaci i przedmiotów, dostępne wartości funkcji atrybutowania) zostały opracowane przez zespół projektantów świata. Nie został stworzony osobny zespół grafików, studenci w misjach wykorzystywali grafiki przygotowane przez studentów z poprzedniego roku, co najwyżej dodając szczegóły stroju czy wyglądu dla rozróżnienia i dopasowania do aktualnie pełnionej funkcji.

Po wstępnym naszkicowaniu przebiegu misji poszczególne grupy projektowe uzgadniały elementy łącznikowe pomiędzy misjami. Zależności pomiędzy misjami ilustruje rysunek 47.

Zespół projektantów świata przygotował dla każdej grupy roboczy świat, zgodny z wymaganiami warunków wstępnych misji. Grupy projektowe przygotowały produkcje szczegółowe a także uzupełniały listę produkcji generycznych tam, gdzie odkryto braki, lub produkcję szczegółową rozbito na hierarchię produkcji o różnym poziomie szczegółowości. Efektem prac każdej grupy była lista produkcji węzłowych danej misji, diagram projektowy z zaznaczonymi liniami fabularnymi, w szczególności jedną wyróżnioną jako główną, pokazujący także wykorzystanie produkcji generycznych pomiędzy produkcjami węzłowymi oraz drzewo hierarchii produkcji danej misji zbudowane na bazie drzewa hierarchii produkcji generycznych.



Rys. 47. Schemat zależności między misjami. Krawędź ciągła oznacza wymagalność efektu misji źródłowej krawędzi w misji docelowej krawędzi, linia przerywana oznacza, że dany efekt jest jednym z alternatywnych wymagań. Symbol gotówki oznacza, że wykonanie misji jest wynagradzane finansowo, a kłódki, że może wiązać się z aresztowaniem.

Korzystając ze skryptu testowania linii fabularnych (elementu implementacji referencyjnej StoryGraph) zespół testował wykonalność zaprojektowanych linii fabularnych i sprawdzał, czy nie otwierają się przed graczem nieplanowane skróty fabularne (ang. *exploit*) umożliwiające przedwczesne dojście do końca misji.

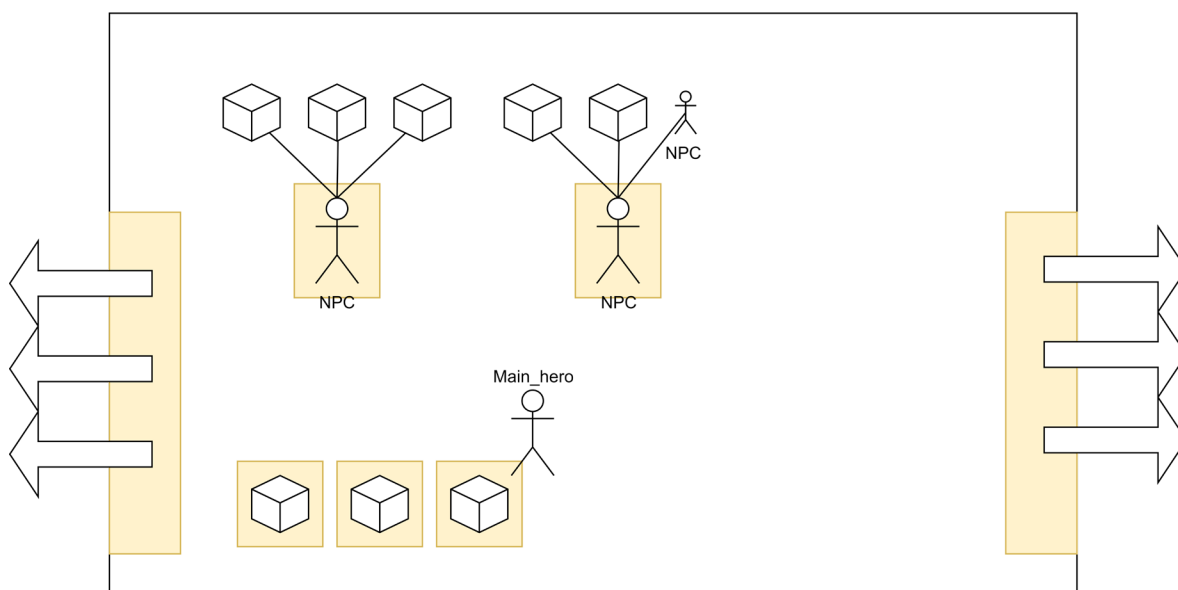
Ostatnim etapem prac było stworzenie pełnego świata rozgrywki, zawierającego wszystkie postacie NPC-ów i elementy pomocnicze, ale w odróżnieniu od światów roboczych, niezawierające żadnych elementów, które mają się pojawić w świecie jako efekty poszczególnych misji. Zbiory produkcji wszystkich misji zostały połączone.

Diagramy projektowe przykładowych misji: początkowej, środkowej i końcowej znajdują się w dodatku D.

6.3 Gry oparte o system StoryGraph

System StoryGraph może być wykorzystany także w silniku fabularnym gry przygodowej. Zwróćmy uwagę, że czego innego oczekuje od aplikacji tester linii fabularnych a czego innego gracz. Gracz nie odtwarza przygotowanej wcześniej linii fabularnej, tylko spontanicznie każdorazowo rozstrzyga, co chciałby zrobić w następnym kroku. Znacznie bardziej musimy dbać o jego immersję, sprowadzając decyzję do jednokrotnego wyboru a nie dwuetapowego dopytywania o wybór akcji i wariantu. Musimy także zadbać o efekt suspense, nie możemy wyświetlać graczowi informacji o możliwym działaniu wraz z opisem konsekwencji („Co robi główny bohater? Bierze udział w korridzie i ginie stratowany przez byka” – to opis przydatny dla testera, ale nie dla gracza).

Rozszerzeniem możliwości systemu w grze jest wykorzystanie zależności przestrzennych przy prezentowaniu opcji wyboru. W samym systemie fabularnym nie rozważamy w ogóle pojęcia odległości między obiektami znajdującymi się w tej samej lokacji. W grze, która jest rozplanowana na przestrzeni ekranu, jak najbardziej możemy uwzględniać zależności przestrzenne i np. filtrować dostępne produkcje i pokazywać te związane z obiektami, obok których główny bohater się znajduje (por. rys. 48, 49). Bo też jednym z największych wyzwań dla projektanta gry wideo jest zaproponowanie interfejsu, w którym wszystkie możliwości gracza są łatwo dostępne i nie wymagają działań wytrącających go z immersji.



Rys. 48. Projekt widoku sceny gry fabularnej z zaznaczonymi obszarami reaktywnymi pozwalającymi na kontekstowe wyświetlanie wyboru produkcji.



Rys. 49. Projekt interfejsu gry fabularnej przygotowany przez studentów.

*

W momencie składania niniejszej rozprawy trwają prace nad dwoma grami opartymi o silnik StoryGraph. Pierwszą z nich jest tworzona dla celów naukowych gra poważna (ang. *serious game*) służąca nabywaniu przez gracza umiejętności z zakresu BHP i ewaluacji tych umiejętności z użyciem algorytmu *Bayesian Knowledge Tracking* zaimplementowanego w oparciu o sekwencje produkcji grafowych wykonywanych przez gracza. Metoda i opis jej stosowania jest zamieszczona w (Grabska-Gradzińska, Argasiński, 2021).

Drugą powstającą grą jest projekt magisterski studenta Informatyki gier komputerowych, który korzysta z systemu StoryGraph poprzez REST API i implementuje typową fabularną grę przygodową z możliwością rozbudowy jej o wgrywanie misji w postaci plików JSON.

6.4 Plany rozwoju projektu

Planowane jest stworzenie modułu analizy afektywnej użytkownika (okulografia oraz reakcja galwaniczna skóry) do badań na użytkownikach w trakcie rozgrywki gry fabularnej opartej o system StoryGraph. Umożliwi to systemowe połączenie znaczników czasowych z danych sensorycznych z etapami rozgrywki i decyzjami podejmowanymi przez gracza a także poszukiwanie wzorców zachowań w kontekście sekwencji wydarzeń fabularnych.

Planowane jest także wykorzystanie silnika StoryGraph do stworzenia komercyjnej gry fabularnej, w oparciu o zawodowe doświadczenia członków zespołu, deweloperów gier wideo.

Podstawowe kierunki rozwoju modelu będą obejmowały:

- Poszerzenie możliwości personalizacji rozgrywki obejmujące zwiększanie otwartości świata, zbieranie informacji o graczach i jego dotychczasowych zachowaniach. co może umożliwić zastosowanie w rozgrywce pętli afektywnej i wprowadzić mechanizmy zapobiegające powtarzalności wybieranych przez gracza rozwiązań.
- Zwiększanie elastyczności rozwoju fabuły poprzez rozbudowanie zależności i powiązań między elementami warstwy narracyjnej, umożliwienie dynamicznej zmiany celów w zależności od rozwoju ścieżki fabularnej lub informacji zwrotnej z modułu analitycznego działającego w tle. Poszerzone zostaną możliwości zmiany roli postaci – patrzenie na fabułę z punktu widzenia różnych postaci w świecie gry.
- Stworzenie mechanizmów automatycznego i proceduralnego generowania sekwencji fabularnych pomiędzy wyznaczonymi przez projektanta punktami węzłowymi. Będzie

to duże wyzwanie pod względem zachowania spójności i logiki związków przyczynowo-skutkowych a także unikania efektu wrażenia niesamowitości pozornie poprawnych ścieżek przy potencjalnych sekwencjach generowanych (jako uogólnienia znanego w literaturze pojęcia *uncanny valley*). Ograniczenie tego efektu będzie wymagało rozbudowy mechanizmów uzależniania dostępności produkcji od kontekstu możliwych produkcji i historii działań już wykonanych w sposób wykraczający poza prosty mechanizm przysyłania produkcji zaimplementowany w niniejszej pracy i będzie dużym wyzwaniem. Dodatkowym wyzwaniem będzie umożliwienie wykorzystanie w procesach automatyzacji sztucznej inteligencji i uczenia maszynowego.

- Dodanie kolejnych narzędzi analitycznych związanych z hierarchią produkcji i poszerzaniem możliwości wyboru ścieżek fabularnych po rozbudowie świata bądź zbioru produkcji a także śledzeniem sekwencji decyzji gracza czy to w kontekście procesu decyzyjnego, immersji i wykorzystania ram fabularnych.

Podsumowanie

Przedstawiona rozprawa doktorska prezentuje model grafowy struktur fabularnych gier wideo i jego wykorzystanie w narzędziach wspomagania kreatywności projektanta gier wideo oraz możliwości, jakie daje zapis i analiza rozgrywki gracza oraz podejmowanych przez niego decyzji fabularnych. Model dostosowany jest do specyfiki struktur narracyjnych, znajduje to odzworowanie w szczególności w autorskiej koncepcji grafu snopkowego stworzonej przez autorkę rozprawy w celu łatwego reprezentowania zależności pomiędzy poszczególnymi obiektami gry w ramach jednej lokacji w grze.

Formalnie zdefiniowane funkcje operujące na grafach w szczególności dopasowanie generyczne i produkcja generyczna oparte o funkcję częściowego etykietowania pozwalają na odwzorowanie w modelu formalnym koncepcji analitycznych zaczerpniętych z teorii literatury i umożliwiają praktyczną realizację ideału dualności autorstwa fabuły współtworzonej przez projektanta gry i użytkownika końcowego. Model został przedstawiony w rozdziale trzecim. W pracy przedstawiono zarówno związki proponowanego modelu z teoriami literaturoznawczymi (w rozdziale pierwszym) jak i konsekwencje jakie zaproponowane struktury formalne niosą dla rozbudowy możliwości fabularnych nieliniowej fabuły growej (w rozdziale czwartym).

Proces rozgrywki dzięki odwzorowaniu jej w postaci sekwencji produkcji grafowych staje się procesem twórczym, pozwala na zachowanie, odtworzenie i późniejsze wykorzystanie jako

podstawy do działań analitycznych i stanowi informację zwrotną dla projektantów o sposobie wykorzystania ram fabularnych gry do tworzenia spersonalizowanej historii wykorzystującej zaprojektowane mechanizmy w sposób zindywidualizowany, zależny od osobistych preferencji i doświadczeń gracza. Problematyka ta została poruszona w rozdziale 6.2.

Ewaluacja jest niezwykle istotnym etapem wytwarzania gry, a ewaluacja spójności fabuły i zaangażowania gracza w tworzenie rozwiązań fabularnych jest niezwykle istotnym elementem procesu ewaluacji. Oparcie budowania narracji o model grafowy pozwala wbudować w proces tworzenia produkcje wymuszające urozmaicenie równoległych ścieżek fabularnych w celu zapobieżenia znudzeniu gracza.

Sformalizowanie i hierarchizacja całości narracyjnych pozwala na wydajną współpracę grup projektowych i podział prac projektowych między członków zespołu. W przyszłych zastosowaniach systemu część prac może być zastąpiona proceduralnym generowaniem fragmentów fabuły opartym o ścieżki wynikające z ograniczeń i zależności zawartych w modelu. Spójność i logiczne powiązania między decyzjami projektowymi dokonywanymi na różnych poziomach abstrakcji i w różnych momentach procesu projektowania zapewniana jest przez mechanizm hierarchizacji i przesłaniania produkcji przedstawione w rozdziale 6.1. Nie rozwiązuje to w pełni problemów nienaturalności i braku zaskoczenia kunsztownymi zwrotami akcji projektowanymi w głównych wątkach fabularnych utworów, ale zmniejsza ryzyko pojawienia się podstawowych błędów prawdopodobieństwa powstającej narracji, może też ograniczyć efekt niesamowitości pozornie poprawnych ścieżek przy potencjalnych sekwencjach generowanych (jako uogólnienia znanego w literaturze pojęcia *uncanny valley*).

Model przeznaczony jest do wykorzystania w dowolnych implementacjach zarówno związanych z procesem projektowym jak i ewaluacją gier przygodowych a także gier poważnych opartych na mechanizmach gier przygodowych. Aby zapewnić jak największą elastyczność dopasowania projektu do różnych zastosowań przedstawiono formalną specyfikację systemu StoryGraph zbudowaną na bazie formalnego modelu grafowego (rozdział 4). Wymusza ona zgodność implementacji z założeniami modelu, ale pozostawia programistom swobodę w doborze języków i narzędzi programowania. Pozwala na rozbudowę implementacji związaną z konkretnymi zastosowaniami projektowanej gry.

Przedstawiona w pracy implementacja (rozdział 5) ma charakter referencyjny. Zaprezentowane są algorytmy pozwalające na praktyczne zastosowanie funkcji dopasowania generycznego, wykonywanie produkcji na zdefiniowanym grafie świata oraz mechanizm przesłaniania

produkcji oparty o własności modelu warstwowego i wskazane jego zastosowania do usprawnienia procesu decyzyjnego gracza. Przedstawiony jest w rozdziale 6.1 algorytm budowania hierarchicznej struktury drzewa produkcji.

Implementacja referencyjna umożliwiła weryfikację tez postawionych w pracy i stała się podstawą do prac implementacyjnych zarówno zespołu badawczego, wykorzystującego system jako podstawę eksperymentów z dziedziny informatyki afektywnej jak i prac edukacyjnych, w tym magisterskich. Prace zespołu projektowego grup studenckich i ich gra „Royal Envoy” stała się podstawą przykładów praktycznych przedstawionych w pracy. Umożliwiła też zebranie bogatego materiału w postaci kilkudziesięciu misji.

Rozwój przedstawionego w niniejszej rozprawie modelu grafowego planowany jest w najbliższej przyszłości w oparciu o grę fabularną i grę poważną. Pozwoli to na przetestowanie koncepcji rozszerzających jego możliwości, które na obecnym etapie pozostają w fazie spekulacji lub na etapie wczesnych testów, czyli możliwość zaprojektowania rozgrywki umożliwiającej zmianę ról w trakcie rozgrywki oraz dynamicznej zmiany celów w zależności od podejmowanych przez gracza decyzji i ujawnianych preferencji. Zbieranie informacji o graczu i jego dotychczasowych zachowaniach może umożliwić także zastosowanie w rozgrywce pętli afektywnej i zapobiec powtarzalności wybieranych przez gracza rozwiązań. Wyzwaniem będzie także zaproponowanie mechanizmów generacyjnych dla całości lub fragmentów misji.

Bibliografia

Teoria literatury i badanie gier

- Aarseth, E. J., *Cybertekst : spojrzenia na literaturę ergodyczną*, Pisarski, M., Schreiber, P., Sikora, D., Tabaczyński, M. (tłum.), Korporacja Ha!art. (2014)
- Aarseth, E., Smedstad M. S., Sunnanå L., *A multidimensional typology of games*, [w:] *DiGRA Conference*, Utrecht University and Digital Games Research Association. (2003)
- Adams, E., *Projektowanie gier, Podstawy*, Janusz, J., Pilch, P., Wieloch, M. (tłum.), Helion, (2011)
- Apperley T. H., *Genre and game studies: Toward a critical approach to video game genres*, „Simulation & Gaming”, 2006; 37(1), s. 6–23. (2006)
- Burzyńska, A., Markowski, M. P., *Teorie literatury XX wieku*, Społeczny Instytut Wydawniczy Znak. (2006).
- Lévi-Strauss, C., *Analiza morfologiczna bajki rosyjskiej*, „Pamiętnik Literacki” 1968 (r. LIX) nr 4, s. 267–284. (1968)
- Majkowski, T., *Język gropowieści*, Wydawnictwo Uniwersytetu Jagiellońskiego. (2019)
- Murray, J. H., *Hamlet on the Holodeck: The Future of Narrative in Cyberspace*, The MIT Press. (2005)
- Propp, W., *Morfologia bajki*, „Pamiętnik Literacki” 1968 (r. LIX) nr 4, s. 203–242. (1968)
- Propp, W., *Morfologia bajki magicznej*, Rojek, P. (tłum.), Zakład Wydawniczy „Nomos”. (2011)
- Propp, V., *Morphology of the Folktale*, wyd. II., Scott, L. (tłum.), University of Texas Press, Austin. (1968)
- Sicart, M., *Defining Game Mechanics*, „Game Studies”, 2008, 8(2). (2008)

Teoria grafów i jej zastosowania

- Biermann, E. et al., *Parallel Independence of Amalgamated Graph Transformations Applied to Model Transformation*. [w:] Engels, G. et al. (red.) *Graph Transformations and Model-Driven Engineering. Lecture Notes in Computer Science*, t. 5765. s. 121–140, Springer, Berlin, Heidelberg. (2010)
- Busatto, G., *An abstract model of hierarchical graphs and hierarchical graph transformation*, s. 1–227, University of Paderborn, Germany. (2002)
- Chitturi, B., Balachander, S., Satheesh, S., Puthiyoppil, K., *Layered Graphs: Applications and Algorithms*, „Algorithms”, 11, 7, (June 2018) s. 93. (2018)
- Chitturi, B., Srinath, T., *Hard Problems on Layered Graphs: Parallel Algorithms and Improvements*, 2018 IEEE 8th International Advance Computing Conference (IACC), Greater Noida, India, s. 207–212. (2018)
- Ehrig, H., Habel, A., *Graph Grammars with Application Conditions*. [w:] *The Book of L*. Springer, Berlin, Heidelberg. (1986)
- Hearn, R. A., *Games, Puzzles, and Computation*. [rozprawa doktorska, MIT]. (2006)
- Hoffmann, B., *Graph Transformation with Variables*. [w:] Kreowski, H. J. et al. (red.) *Formal Methods in Software and Systems Modeling. Lecture Notes in Computer Science*, t. 3393. Springer, Berlin. (2005)
- Kreowski, H.-J., Kuske, S., Toennies, H., *Autonomous Units to Model Games*, [w:] *GI Jahrestagung*, s. 3465–3472. (2009)
- Morales, L. P., Esteban, A. D., Gervás, P., *Concept-graph based biomedical automatic summarization using ontologies*, [w:] *Proceedings of the 3rd Textgraphs Workshop on Graph-Based Algorithms for Natural Language Processing (TextGraphs-3)*. Association for Computational Linguistics, USA, 53–56. (2008)
- Padberg, J., *Hierarchical Graph Transformation Revisited – Transformations of Coalgebraic Graphs*, [w:] de Lara, J., Plump, D. (red.) *Graph Transformation. ICGT 2017. Lecture Notes in Computer Science*, t. 10373. Springer, Cham. (2017)
- Palacz, W., *Algebraic hierarchical graph transformation*, „J. Comput. Syst. Sci.” 68(3): 497–520. (2004)

- Palacz, W., *Hierarchical graph transformations with meta-rules*, „Annales Universitatis Mariae Curie-Sklodowska sectio AI – Informatica” 8(1):15–21. (2008)
- Palacz, W., Ryszka, I., Grabska, E., *Graphs with layers: a visual tool for conceptual design and graph generation*, [w:] Li, H., Rafiq, Y., de Wilde, P. (red.), *Proceedings of the 21st International Workshop: Intelligent Computing in Engineering (EG-ICE 2014), July 16–18, 2014 in Cardiff, United Kingdom*, s. 1–11. Cardiff University School of Engineering. (2014).
- Plaza, L., Díaz, A., Gervás, P., *A semantic graph-based approach to biomedical summarisation*. „Artif. Intell. Med.” 53, 1 (September, 2011), s. 1–14. (2011)
- Priemer, D., George, T., Hahn, M., Raesch, L., Zündorf, A.: *Using Graph Transformation for Puzzle Game Level Generation and Validation*. [w:] Echahed, R., Minas, M. (red.) *Graph Transformation. ICGT 2016. Lecture Notes in Computer Science*, t. 9761. Springer, Cham. (2016)
- Rensink, A., Kuperus, J. H., *Repotting the Geraniums: On Nested Graph Transformation Rules*, „Electron. Commun. Eur. Assoc. Softw. Sci. Technol.” 18. (2009)
- Rozenberg, G., *Handbook of Graph Grammars and Computing by Graph Transformations*, t. 1: *Foundations*. (1997)
- Sanchez, D., Florez, H., *Improving Game Modeling for the Quoridor Game State Using Graph Databases*. [w:] *Proceedings of the International Conference on Information Technology & Systems (ICITS 2018). ICITS 2018. Advances in Intelligent Systems and Computing*, t. 721, s. 333–342. Springer, Cham. (2018)
- Sato, M., Anada, K., Tsutsumi, M., *Formulations of patterns by a graph model for the game of Go*. „Journal of Computational Methods in Sciences and Engineering”, 17, S1, (January 2017) s. S111–S121. (2017)
- Schürr, A., *Progress: A VHL-language based on graph grammars*, [w:] Ehrig, H., Kreowski, H.-J., Rozenberg, G. (red.) *Graph Grammars and Their Application to Computer Science. Graph Grammars 1990. Lecture Notes in Computer Science*, t. 532. Springer, Berlin, Heidelberg. (1991)
- Strug, B., Paszyńska, A., Paszyński, M., Grabska, E., *Using a graph grammar system in the finite element method*, „International Journal of Applied Mathematics and Computer Science”, 23, 4 (December 2013), s. 839–853. (2013)

Strug, B., Ślusarczyk, G., Paszyńska, A., Palacz, W., *A survey of different graph structures used in modeling design, engineering and computer science problems*, [w:] Zawislak, S., Rysiński, J. (red.), *Graph-based modelling in science, technology and art*, s. 243–275, Springer International Publishing. (2022)

Ślusarczyk, G., *Graph-based representation of design properties in creating building floor-plans*. *Computer Aided Design* 95, (February 2018), 24–39. (2018)

Prace opisujące system StoryGraph i obszary jego zastosowań

Argasiński, J., Grabska-Gradzińska, I., *Applying layered graphs to evidence centered design based serious games assessment*, [w:] Sampson, D. G., Ifenthaler, D., Isaías, P. (red.), *Proceedings of the 17th International Conference on Cognition and Exploratory Learning in the Digital Age (Celda 2020): November 18–20, 2020*, s. 365–369. IADIS Press (2020).

Argasiński, J., Grabska-Gradzińska, I., *Patterns in serious game design and evaluation application of eye-tracker and biosensors*, [w:] Rutkowski, L. et al. (red.), *Artificial Intelligence and Soft Computing: 16th International Conference, ICAISC 2017, Zakopane, Poland, June 11–15, 2017: proceedings*, t. 2, s. 367–377). Springer International Publishing. (2017)

Grabska, E., Grabska-Gradzińska, I., *Towards modeling structures of the game worlds by systems of graphs*, [w:] *Proceedings of International Conference on Engineering and Emerging Technologies (ICEET '21)*, Institute of Electrical and Electronics Engineers (IEEE). (2021)

Grabska-Gradzińska, I., Argasiński, J., *Graph-based method for the interpretation of user activities in serious games*, [w:] Ardito, C. et al. (red.), *Human-Computer Interaction – INTERACT 2021: 18th IFIP TC 13 International Conference Bari, Italy, August 30 – September 3, 2021, proceedings*, t. 3, s. 87–96, Springer Nature. (2021)

Grabska-Gradzińska, I., Argasiński, J., *Serious games evaluation using eye-tracking and affective computing techniques case study of OSH training course.*, [w:] Oberhauser, R. et al. (red.), *ACHI 2017: the Tenth International Conference on Advances in Computer-Human Interactions: March 19–23, 2017, Nice, France*, s. 260–264. IARIA. (2017).

- Grabska-Gradzińska, I., Argasiński, J., *Patterns in video games analysis: application of eye-tracker and electrodermal activity (EDA) sensor*. [w:] Rutkowski, L. et al. (red.), *Artificial Intelligence and Soft Computing: 17th International Conference, ICAISC 2018, Zakopane, Poland, June 3-7, 2018: proceedings*, t. 2, s. 619–629. Springer. (2018).
- Grabska-Gradzińska, I., Grabska, E., Nowak, L., Palacz, W., *Towards Automatic Generation of Storyline Aided by Collaborative Creative Design*, [w:] *Cooperative Design, Visualization, and Engineering (CDVE 2020). Lecture Notes in Computer Science*, t. 12341, s. 47–56, Springer, Cham. (2020)
- Grabska-Gradzińska, I., *How to predict behavior at risk: educational game story analysis using layered graphs and eyetracking system*, [w:] Ślusarczyk, G., Strug, B., de Wilde, P. (red.), *Proceedings of the 23rd International Workshop on Intelligent Computing in Engineering: EG-ICE 2016: 29 June – 1 July Kraków, Poland*, s. 75–84. (2016).
- Grabska-Gradzińska, I., Nowak, L., Grabska, E., *Automatic story generation based on graph model using Godot Engine*. [w:] Rutkowski, L. et al. (red.), *Artificial Intelligence and Soft Computing: 19th International Conference, ICAISC 2020, Zakopane, Poland, October 12-14, 2020: proceedings*, cz. I, s. 397–405. Springer. (2020).
- Grabska-Gradzińska, I., Nowak, L., Palacz, W., Grabska, E., *Application of graphs for story generation in video games*. [w:] Stanger, N., Joachim, V. L. (red.), *Proceedings of the Australasian Computer Science Week Multiconference 2021 (ACSW 2021): 1st–5th February 2021, University of Otago, Dunedin, New Zealand (Virtual)*, The Association for Computing Machinery. (2021)
- Grabska-Gradzińska, I., Nowak, L., Palacz, W., Grabska, E., *Collaborative Story Generation using JSON in Godot Engine as Decision Making Support Tool*, [w:] *International Conference on Cognitive Decision Support Systems & Technologies Proceedings (ICDSST 2020). 27–29 May 2020, University of Zaragoza, Zaragoza, Spain*. (2020)
- Grabska-Gradzińska, I., Porębski, B., Palacz, W., Grabska, E. *Graph-based data structures of computer games*, *Annual International Conference on Computer Games, Multimedia & Allied Technology*, s. 88–96. (2013).
- Nowak, L., Grabska-Gradzińska, I., Grabska, E., Palacz, W., Wrona, M., Konopka, A., Mańka, K., Okrzesik, M., Urban, D., Szypura, K., Mikołajczyk, A., Kuligowicz, J., *Graph rules hierarchy as a tool of collaborative game narration creation*, [w:] Luo, Y.,

(red.), *Cooperative Design, Visualization, and Engineering. CDVE 2021. Lecture Notes in Computer Science*, t. 12983. Springer, Cham. (2021)

Palacz, W., Grabska-Gradzińska, I., Nowak, L., Grabska, E., *Designing Plots for Multiplayer Games with the Use of Graph Transformation Rules*, [w:] Luo, Y. (red.) *Cooperative Design, Visualization, and Engineering. CDVE 2021. Lecture Notes in Computer Science*, t. 12983. Springer, Cham. (2021)

Palacz, W., Grabska-Gradzińska, I., *Textual representation of pushout transformation rules*, [w:] Luo, Y. (red.), *Cooperative Design, Visualization, and Engineering: 17th International Conference, CDVE 2020, Bangkok, Thailand, October 25–28, 2020, proceedings*, s. 75–80. Springer. (2020)

Generowanie fabuł i ich opisów słownych

Adams, T., *Secret identities in dwarf fortress*, [w:] *Proceedings Experimental AI in Games (AIIDE 2017)*, s. 22–25. (2017)

Adams, T., *Simulation principles from Dwarf Fortress*. [w:] *Game AI Pro 2: Collected Wisdom of Game AI Professionals*, s. 519–521. CRC Press. (2017)

Albornoz de, J. C., Plaza, L., Gervás, P., Díaz, A., *A joint model of feature mining and sentiment analysis for product review rating*, [w:] *Proceedings of the 33rd European conference on Advances in information retrieval (ECIR '11)*. Springer-Verlag, Berlin, Heidelberg, s. 55–66. (2011)

Albornoz de, J.C., Plaza, L., Gervás, P., *A hybrid approach to emotional sentence polarity and intensity classification*, [w:] *Proceedings of the Fourteenth Conference on Computational Natural Language Learning (CoNLL '10)*. Association for Computational Linguistics, USA, s. 153–161. (2010)

Alves, A. et al., *Conceptual enrichment of locations pointed out by the user*, [w:] *Proceedings of the 11th international conference, KES 2007 and XVII Italian workshop on neural networks conference on Knowledge-based intelligent information and engineering systems: Part I (KES '07/WIRN '07)*. Springer-Verlag, Berlin, Heidelberg, s. 346–353. (2007)

Ballesteros, M. et al., *UCM-2: a rule-based approach to infer the scope of negation via dependency parsing*, [w:] *Proceedings of the First Joint Conference on Lexical and Computational Semantics*, t. 1: *Proceedings of the main conference and the shared task*, t. 2:

- Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval '12)*. Association for Computational Linguistics, USA, s. 288–293. (2012)
- Ballesteros, M. et al., *Towards an n-version dependency parser*, [w:] *Proceedings of the 13th international conference on Text, speech and dialogue (TSD '10)*. Springer-Verlag, Berlin, Heidelberg, s. 43–50. (2010)
- Bautista, S., Gervás, P., Díaz, A., *Adaptive text generation based on emotional lexical choice*, [w:] *Proceedings of the XV International Conference on Human Computer Interaction (Interacción '14)*. Association for Computing Machinery, New York, NY, USA, Article 20, 1–2. (2014)
- Beltrán-Ferruz, P. J., González-Calero, P. A., Gervás, P., *Converting Mikrokosmos frames into description logics*, [w:] *Proceedings of the Workshop on NLP and XML (NLPXML-2004): RDF/RDFS and OWL in Language Technology (NLPXML '04)*. Association for Computational Linguistics, USA, s. 35–42. (2004)
- Buss, D. B., Eland, M. V., Lystlund, R., Burelli, P., *The Quality System – An Attempt to Increase Cohesiveness Between Quest Givers and Quest Types*. [w:] *Interactive Storytelling (ICIDS 2015)*. Lecture Notes in Computer Science, t. 9445. Springer, Cham. (2015)
- Compton, K., Filstrup, B., Mateas, M., *Tracery: Approachable story grammar authoring for casual users*. [w:] *Papers from Seventh Intelligent Narrative Technologies Workshop (INT 07)*, s. 63–67. (2007)
- Compton, K., Kybartas, B., Mateas, M., *Tracery: an author-focused generative text tool*. [w:] *Interactive Storytelling (ICIDS 2015)*. Lecture Notes in Computer Science, t. 9445, Springer, Cham. (2015)
- Concepción, E., Gervásand, P., Méndez, G., *Afanasyev: A collaborative architectural model for automatic story generation*, [w:] *Proceedings of 5th AISB Symposium on Computational Creativity*. (2018)
- Concepción, E. et al., *Using CNL for Knowledge Elicitation and Exchange Across Story Generation Systems*, [w:] *Proceedings of the 5th International Workshop on Controlled Natural Language*, t. 9767 (CNL 2016). Springer-Verlag, Berlin, Heidelberg, s. 81–91. (2016)

- Cosimano, M., *CD Projekt Red designer breaks down The Witcher 3's best quest*, <https://www.destructoid.com/stories/cd-projekt-red-designer-breaks-down-the-witcher-3-s-best-quest-438115.phtml> (2017, data dost.: 9 XI 2020)
- Delatorre, P. et al., *A computational model of the cognitive impact of decorative elements on the perception of suspense*. „Connect. Sci” 29, 4 (December 2017), s. 295–331. (2017)
- Díaz, A., Gervás, P., *User-model based personalized summarization*. „Inf. Process. Manage.” 43, 6 (November, 2007), s. 1715–1734. (2007)
- Díaz-Agudo, B., Gervás, P., González-Calero, A. P., *Adaptation guided retrieval based on formal concept analysis*, [w:] *Proceedings of the 5th international conference on Case-based reasoning: Research and Development (ICCBR '03)*. Springer-Verlag, Berlin, Heidelberg, 131–145. (2003)
- Díaz-Agudo, B., Gervás, P., González-Calero, A. P., *Poetry Generation in COLIBRI*, [w:] *Proceedings of the 6th European Conference on Advances in Case-Based Reasoning (ECCBR '02)*. Springer-Verlag, Berlin, Heidelberg, s. 73–102. (2002)
- Dionne, D. et al., *A model for human readable instruction generation using level-based discourse planning and dynamic inference of attributes disambiguation*, [w:] *Proceedings of the 12th European Workshop on Natural Language Generation (ENLG '09)*. Association for Computational Linguistics, USA, s. 66–73. (2009)
- Doran, J., Parberry, I., *A prototype quest generator based on a structural analysis of quests from four MMORPGs*, [w:] *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games (PCGames '11)*. Association for Computing Machinery, New York, NY, USA, s. 1–8. (2011)
- Francisco, V. et al., *EmoTales: creating a corpus of folk tales with emotional annotations*. „Lang. Resour. Eval.” 46, 3 (September 2012), s. 341–381. (2012)
- Francisco, V., Gervás, P., *Automated mark up of affective information in english texts*, [w:] *Proceedings of the 9th international conference on Text, Speech and Dialogue (TSD'06)*. Springer-Verlag, Berlin, Heidelberg, 375–382. (2006)
- Francisco, V., Gervás, P., Peinado, F., *Ontological reasoning for improving the treatment of emotions in text*. „Knowl. Inf. Syst.” 25, 3 (December 2010), s. 421–443. (2010)

- García, R. et al., *A framework for the e-r computational creativity model*, [w:] *Proceedings of the 5th Mexican international conference on Artificial Intelligence (MICAI'06)*. Springer-Verlag, Berlin, Heidelberg, s. 70–80. (2006)
- García-Ortega, R. H. et al., *How the World Was MADE: Parametrization of Evolved Agent-Based Models for Backstory Generation*. [w:] *Applications of Evolutionary Computation (EvoApplications 2015)*. *Lecture Notes in Computer Science*, t. 9028. Springer, Cham. (2011)
- García-Ortega, R. H., et al., *The Story of Their Lives: Massive Procedural Generation of Heroes' Journeys Using Evolved Agent-Based Models and Logical Reasoning*. [w:] *Applications of Evolutionary Computation (EvoApplications 2016)*. *Lecture Notes in Computer Science*, t. 9597. Springer, Cham. (2016)
- Gervás, P. et al., *The long path to narrative generation*. „IBM J. Res. Dev.” 63, 1, Article 1 (January/February 2019). (2019)
- Gervás, P. et al., *Integrating aggregation strategies in an in-home domain dialogue system*, [w:] *Proceedings of the 13th international conference on Text, speech and dialogue (TSD '10)*. Springer-Verlag, Berlin, Heidelberg, s. 499–506. (2010)
- Gervás, P. et al., *Story plot generation based on CBR*. „Know.-Based Syst.” 18, 4–5 (August, 2005), s. 235–242. (2005)
- Gervás, P., *Constrained creation of poetic forms during theme-driven exploration of a domain defined by an N-gram model*. „Connect. Sci” 28, 2 (June 2016), s. 111–130. (2016)
- Gervás, P., *Deconstructing Computer Poets: Making Selected Processes Available as Services*. „Comput. Intell.” 33, 1 (February 2017), s. 3–31. (2017)
- Gervás, P., *Engineering linguistic creativity: bird flight and jet planes*, [w:] *Proceedings of the NAACL HLT 2010 Second Workshop on Computational Approaches to Linguistic Creativity (CALC '10)*. Association for Computational Linguistics, USA, s. 23–30. (2010)
- Gervás, P., León, C., *When reflective feedback triggers goal revision: a computational model for literary creativity*, [w:] *Proceedings of the First International Conference on AI and Feedback*, t. 1407 (AInF '15). CEUR-WS.org, Aachen, DEU, s. 32–39. (2015)
- Gervás, P., *Modeling Literary Style for Semi-automatic Generation of Poetry*, [w:] *Proceedings of the 8th International Conference on User Modeling 2001 (UM '01)*. Springer-Verlag, Berlin, Heidelberg, s. 231–233. (2001)

- Gervás, P., *UCM submission to the surface realization challenge*, [w:] *Proceedings of the 13th European Workshop on Natural Language Generation (ENLG '11)*. Association for Computational Linguistics, USA, s. 239–241. (2011)
- Grinblat, J., Bucklew, C. B., *Subverting historical cause & effect: generation of mythic biographies in Caves of Qud*, [w:] *Proceedings of the 12th International Conference on the Foundations of Digital Games (FDG '17)*. Association for Computing Machinery, New York, NY, USA, Article 76, 1–7. (2017)
- Grinblat, J., Bucklew, C. B., *Warm Rocks for Cold Lizards: Generating Meaningful Quests in Caves of Qud*, [w:] *Experimental AI in Games Workshop (AIIDE 2020)*. (2020)
- Hagström, A., *Poetically Man Dwells in Game Space: A Phenomenological Investigation of Video Games as Art*, rozprawa doktorska, Uppsala University, Disciplinary Domain of Humanities and Social Sciences, Faculty of Arts, Department of Game Design. (2017)
- Hervás, R. et al., *Enrichment of automatically generated texts using metaphor*, [w:] *Proceedings of the artificial intelligence 6th Mexican international conference on Advances in artificial intelligence (MICAI '07)*. Springer-Verlag, Berlin, Heidelberg, s. 944–954. (2007)
- Hervás, R., Gervás, P., *Agent-Based solutions for natural language generation tasks*, [w:] *Proceedings of the 11th Spanish association conference on Current Topics in Artificial Intelligence (CAEPIA '05)*. Springer-Verlag, Berlin, Heidelberg, 103–112. (2005)
- Hervás, R., Gervás, P., *Case-based reasoning for knowledge-intensive template selection during text generation*, [w:] *Proceedings of the 8th European conference on Advances in Case-Based Reasoning (ECCBR'06)*. Springer-Verlag, Berlin, Heidelberg, s. 151–165. (2006)
- Khaliq, I., Watson, Z., *The Omni Framework: A Destiny-Driven Solution to Dynamic Quest Generation in Games*, 2018 IEEE Games, Entertainment, Media Conference (GEM), Galway, 2018, s. 306–311. (2018)
- Konert, J., Wendel, V., Göbel, S., Steinmetz, R., *Towards an analysis of cooperative learning-behaviour in social dilemma games*. [w:] *Proceedings of the European Conference on Games-based Learning (ECGBL 2011)*, s. 329–332. (2012)
- León, C. et al., *Mixed Narrative and Dialog Content Planning Based on BDI Agents*, [w:] *Current Topics in Artificial Intelligence: 12th Conference of the Spanish Association*

- for Artificial Intelligence, CAEPIA 2007, Salamanca, Spain, November 12-16, 2007. Selected Papers.* Springer-Verlag, Berlin, Heidelberg, s. 150–159. (2007)
- León, C., Gervás, P., *A top-down design methodology based on causality and chronology for developing assisted story generation systems*, [w:] *Proceedings of the 8th ACM conference on Creativity and cognition (C&C '11)*. Association for Computing Machinery, New York, NY, USA, s. 363–364. (2011)
- León, C., Gervás, P., *The Role of Evaluation-Driven Rejection in the Successful Exploration of a Conceptual Space of Stories*. „Minds Mach.” 20, 4 (November 2010), s. 615–634. (2010)
- Martins, P. et al., *Computational creativity infrastructure for online software composition: a conceptual blending use case*. „IBM J. Res. Dev.” 63, 1, Article 1 (January/February 2019). (2019)
- Mendonça, M. R. F., Ziviani, A., *Network-Based Procedural Story Generation*. „Comput. Entertain.” 16, 3, Article 3. (2018)
- Peinado, F. et al., *Assessing the Novelty of Computer-Generated Narratives Using Empirical Metrics*. „Minds Mach.” 20, 4 (November 2010), s. 565–588. (2010)
- Peinado, F., Gervás, P., *Automatic direction of interactive storytelling: formalizing the game master paradigm*, [w:] *Proceedings of the 4th international conference on Virtual storytelling: using virtual reality technologies for storytelling (ICVS'07)*. Springer-Verlag, Berlin, Heidelberg, s. 196–201. (2007)
- Peinado, F., Gervás, P., *Evaluation of automatic generation of basic stories*. „New Gen. Comput.” 24, 3 (Sep 2006), s. 289–302. (2006)
- Peinado, F., Gervás, P., *Minstrel reloaded: from the magic of lisp to the formal semantics of OWL*, [w:] *Proceedings of the Third international conference on Technologies for Interactive Digital Storytelling and Entertainment (TIDSE'06)*. Springer-Verlag, Berlin, Heidelberg, s. 93–97. (2006)
- Peinado, F., Navarro, Á., Gervás, P., *A testbed environment for interactive storytellers*, [w:] *Proceedings of the 2nd international conference on INtelligent TEchnologies for interactive enterTAINment (INTETAIN '08)*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), Brussels, BEL, Article 3, s. 1–7. (2008)

- Pita, J., Magerko, B., Brodie, S., *True story: dynamically generated, contextually linked quests in persistent systems*. [w:] *Proceedings of the 2007 conference on Future Play (Future Play '07)*, Association for Computing Machinery, New York, NY, USA, s. 145–151. (2007)
- Putzke, J., Fischbach, K., Schoder, D., *Power structure and the evolution of social networks in massively multiplayer online games*, [w:] *18th European Conference on Information Systems (ECIS 2010)*. (2010)
- Riedl, M.O., Young, M. R., *Narrative planning: Balancing plot and character*, „Journal of Artificial Intelligence Research” 39, s. 217–268. (2010)
- Veale, T., Gervás, P., Pérez y Pérez, R., *Computational Creativity: A Continuing Journey*. „Minds Mach.” 20, 4 (November 2010), s. 483–487. (2010)
- Veale, T., Gervás, P., Pease, A., *Understanding creativity: A computational perspective*. „New Gen. Comput.” 24, 3 (Sep 2006), s. 203–207. (2006)
- Xiao, P. et al., *Conceptual Representations for Computational Concept Creation*. „ACM Comput. Surv.” 52, 1, Article 9 (January 2020). (2019)

Technologie

- Articy. [The solution for interactive storytelling and game content management.] (2014). Articy Software GmbH & Co. KG, <https://www.articy.com>, last accessed 2022/03/02
- Creation Kit. [Skyrim editor]. Bethesda Softworks. (2011) <https://www.creationkit.com>, last accessed 2022/03/02
- Ellson, J. et al., *Graphviz and Dynagraph – Static and Dynamic Graph Drawing Tools*, [w:] Jünger, M., Mutzel, P. (red.) *Graph Drawing Software. Mathematics and Visualization*. Springer, Berlin, Heidelberg. (2004)
- Gansner, E. R., Koutsofios, E. North, S., *Drawing graphs with dot*, January 5, 2015
- Godot Engine (3.2) [Game Engine], <https://godotengine.org/>, last accessed 2022/03/02
- Ink. [A narrative scripting language for games.]. inkle (2011) <https://www.inklestudios.com/ink/>, last accessed 2022/03/02
- JSON Schema, <https://json-schema.org/>, last accessed 2022/03/02

Plot Generator. Set of text generator tools (January 2019) Retrieved November 9, 2020 from
<https://www.plot-generator.org.uk/>

Twine. [Open-source tool for telling interactive, nonlinear stories.] (2009). Chris Klimas.
<https://twinery.org/>, last accessed 2022/03/02

Spis gier wymienionych w pracy

W opisach wskazany jest najpierw producent, potem wydawca gry. Jeśli to ten sam podmiot, występuje tylko raz.

Assassin's Creed, Ubisoft Montreal, 2007

Blade Runner, Westwood Studios, Virgin Interactive, 1997

Borderlands, Gearbox Software, 2K Games, 2009

Brothers: A Tale of Two Sons, Starbreeze AB, 505 Games, 2013

Call of Cthulhu, Chaosium, 1981

Call of Cthulhu: Shadow of the Comet, Infogrames, 1993

Call of Cthulhu: Dark Corners of the Earth, Bethesda, 2006

Call of Cthulhu, Cyanide Studio, 2018

Commandos: Za linię wroga, Pyro Studios, Square Enix, 1998

Diablo, Blizzard Entertainment, 1996

Discworld, Perfect 10 Productions, Psygnosis, 1995

Dragon Age, BioWare, Electronic Arts Inc., 2009

Dungeon Master II: The Legend of Skullkeep, FTL Games, Interplay Entertainment, 1995

Fallout, Interplay Entertainment, 1997

Fallout 2, Black Isle Studios, Interplay Entertainment, 1998

Fallout 3, Bethesda Softworks, 2008

Fallout 4, Bethesda Softworks, 2015

Far Cry, Crytek, Ubisoft, 2004

Gamedec, Anshar Studios, 2021

King's Quest, Sierra, 1984

Legend of Grimrock, Almost Human Ltd., 2012

Legend of Grimrock II, Almost Human Ltd., 2014

Lords of the Fallen, CI Games / Deck13 Interactive, Bandai Namco Entertainment, 2014

Star Wars: Galactic Battlegrounds, Ensemble Studios, LucasArts, 2001

Syberia, Microids/Anuman Interactive, 2002

Teenagent, Metropolis Software, Union Logic Software Publishing, Inc., 1994

The Elder Scrolls IV: Oblivion, Bethesda Softworks, 2006

The Elder Scrolls V: Skyrim, Bethesda Softworks, 2011

The Legend of Kyrandia: Hand of Fate, Westwood Studios, Virgin Interactive, 1993

The Longest Journey, Funcom, 2000

The Secret of Monkey Island, LucasArts, 1990

Wiedźmin, CD Projekt RED, Atari / Infogrames, 2007

Wolfenstein, Raven Software, Activision Blizzard, 2009

World of Warcraft, Blizzard Entertainment, 2004

Spis ilustracji

Jeśli nie wskazano inaczej, źródłem jest praca własna autorki.

Rys. 1. Mapa lokacji w systemie gry.	27
Rys. 2. Przykładowe postaci w grze. Źródło: projekt studencki	27
Rys. 3. Przykładowa lokacja. Źródło: projekt studencki.....	28
Rys. 4. Graf stanu świata gry (GWSG) z wyróżnionymi czterema warstwami.	30
Rys. 5. Idea grafu snopkowego.	31
Rys. 6. Podgraf snopkowy (A) oraz półsnopkowy (B) w strukturze GWSG.	32
Rys. 7. Graf lokacji.	34
Rys. 8. Graf postaci.	35
Rys. 9. Graf przedmiotów.	36
Rys. 10. Graf narracji.	36
Rys. 11. Wizualizacja GWSG zbudowanego w <i>Przykładzie 4</i>	41
Rys. 12. GWSG z zaznaczonym podgrafem snopkowym i półsnopkowym.	45
Rys. 13. Przykładowe dane wejściowe algorytmu dopasowania.	71
Rys. 14. Pierwszy krok dopasowania warstwy lokacji.	72
Rys. 15. Drugi krok dopasowania warstwy lokacji.	72
Rys. 16. Trzeci krok dopasowania warstwy lokacji.	73
Rys. 17. Czwarty krok dopasowania warstwy lokacji.	73
Rys. 18. Rekurencyjne zagłębienie algorytmu.	74
Rys. 19. Iloczyn kartezjański wyników funkcji wywołanej rekurencyjnie.....	74
Rys. 20. Podobieństwo lewej strony produkcji bazowej i szczegółowej.	77
Rys. 21. Podobieństwo zbioru instrukcji produkcji bazowej i szczegółowej.	78
Rys. 22. Przeglądarkowy interfejs użytkownika udostępniany przez walidator poprawności produkcji grafowych.	84
Rys. 23. Przykładowy graf świata.	89
Rys. 24. Wizualizacja produkcji interakcji zakończonej zmianą lokacji.	90
Rys. 25. Wizualizacja produkcji walki z ucieczką.....	91
Rys. 26. Wizualizacja produkcji z wartościami wyliczanymi wyrażeniem.	92
Rys. 27. Przykładowa hierarchia produkcji.	96

Rys. 28. Przykładowa hierarchia produkcji z zaznaczonymi błędami.....	96
Rys. 29. Zestaw przykładowych produkcji wraz z zaznaczeniem ich poziomu w hierarchii.....	100
Rys. 30. Fragment drzewa hierarchii produkcji zawierający produkcje o różnych stylach dziedziczenia.	109
Rys. 31. Misja zdobycia naszyjnika oparta o produkcje szczegółowe.....	112
Rys. 32. Drzewo hierarchii produkcji dla misji opartej o produkcje szczegółowe.....	112
Rys. 33. Misja zdobycia naszyjnika oparta o produkcje generyczne.....	112
Rys. 34. Drzewo hierarchii produkcji dla misji opartej o produkcje szczegółowe.....	113
Rys. 35. Schemat współpracy projektantów gry w systemie.	116
Rys. 36. Schemat misji „Dragon Story”.....	119
Rys. 37. Fragment schematu. Przykład produkcji szczegółowej.	119
Rys. 38. Fragment schematu. Przykład produkcji generycznej.	120
Rys. 39. Fragmenty schematu. Zakończenia misji.....	120
Rys. 40. Fragment schematu. Działanie poprzedzające.	120
Rys. 41. Wariant podstawowy wykonania misji uwzględniony w schemacie.....	121
Rys. 42. Wariant trywialny wykonania misji zakończony sukcesem.	122
Rys. 43. Wariant trywialny wykonania misji zakończony klęską.....	122
Rys. 44. Wariant „legalistyczny” wykonania misji.....	123
Rys. 45. Wariant wykonania misji wykorzystujący znajomość legendy.	124
Rys. 46. Drzewo hierarchii produkcji misji „Dragon Story”.	126
Rys. 47. Schemat zależności między misjami.....	128
Rys. 48. Projekt widoku sceny gry fabularnej z zaznaczonymi obszarami reaktywnymi pozwalającymi na kontekstowe wyświetlanie wyboru produkcji.	130
Rys. 49. Projekt interfejsu gry fabularnej przygotowany przez studentów. Źródło: projekt studencki.....	130

Dodatek A – Objaśnienia symboli użytych w definicjach

A – zbiór atrybutów (def. 2)

C – etykietowany graf skierowany (def. 1)

C – procedura sprawdzania predykatów stosowalności (def. 17)

$D = \{D_a\}_{a \in A}$ – rodzina dziedzin atrybutów a ze zbioru A (def. 3)

E – zbiór krawędzi (def. 1)

G – graf atrybutowany (def. 2)

G_I – instancja grafu atrybutowanego (def. 3)

H – graf n -warstwowy (def. 4)

K – graf atrybutowany z częściowym etykietowaniem (def. 11)

K_I – instancja grafu atrybutowanego z częściowym etykietowaniem (def. 12)

L – alfabet etykiet wierzchołków (przed def. 1)

L_H – alfabet etykiet warstwowych (przed def. 4)

M – graf n -warstwowy z częściowym etykietowaniem (def. 13, wykorzystany w def. 16)

N – graf n -warstwowy z częściowym etykietowaniem lub instancja takiego grafu (def. 16)

P – wykorzystywane w definicji ścieżki: t_P (def. 7)

P – procedura modyfikująca przekształcany graf (def. 17)

Q – graf atrybutowany z częściowym etykietowaniem lub instancja takiego grafu (def. 15)

R – wykorzystywane w: x_R – wyróżniona warstwa (def. 8)

V – zbiór wierzchołków grafu (def. 1)

X – zbiór wierzchołków zwanych warstwami (def. 4)

$\Gamma(L, A)$ – zbiór wszystkich skierowanych atrybutowanych grafów etykietowanych z etykietami ze zbioru L i atrybutami ze zbioru A (def. 4)

$\Gamma_I(L, A)$ – zbiór wszystkich instancji grafów z $\Gamma(L, A)$ (def. 4)

$\Delta(L, A)$ – zbiór wszystkich skierowanych atrybutowanych grafów z częściowym bądź pełnym etykietowaniem z etykietami ze zbioru L i atrybutami ze zbioru A (def. 13)

$\Delta_I(L, A)$ – zbiór wszystkich instancji grafów z $\Delta(L, A)$ (def. 13)

a – atrybut (def. 2)

$attr : V \cup E \rightarrow 2^A$ – funkcja atrybutowania (def. 2)

$ch : X \rightarrow \Gamma(L, A)$ – funkcja przypisująca każdej warstwie zagnieżdżony w niej graf (def. 4)
 $dom(lab)$ – zbiór argumentów funkcji częściowej lab lub dziedzina funkcji totalnej lab (def. 11)
 e – krawędź w grafie (def. 7)
 i – licznik wężła w ścieżce (def. 7)
 k – dopasowanie generyczne, czyli para funkcji $k^1 = k_V$ i $k^2 = k_E$ (def. 16)
 k_E – funkcja przekształcająca krawędzie (def. 15)
 k_V – funkcja przekształcająca wierzchołki (def. 15)
 k_x – dopasowanie generyczne z rodziny $\{ k_x \}$ dla $x \in X_M$ (def. 16)
 $lab : V \rightarrow L$ – funkcja etykietowania wierzchołków (def. 1)
 m – długość ścieżki (def. 7)
 m_X – funkcja przekształcająca warstwy (def. 16)
 m_E – funkcja przekształcająca krawędzie (def. 16)
 n – liczba warstw (def. 4)
 p – produkcja generyczna (def. 17)
 $s : E \rightarrow V$ – funkcja źródłowa krawędzi (def. 1)
 $s_P : E \cup \bigcup_{x \in X} E_{ch(x)} \rightarrow \bigcup_{x \in X} V_{ch(x)}$ – jednolita funkcja źródłowa krawędzi, czyli suma funkcji źródłowych występujących w danym grafie warstwowym (def. 7)
 $t : E \rightarrow V$ – funkcja docelowa krawędzi (def. 1)
 $t_P : E \cup \bigcup_{x \in X} E_{ch(x)} \rightarrow \bigcup_{x \in X} V_{ch(x)}$ – jednolita funkcja docelowa krawędzi, czyli suma funkcji docelowych występujących w danym grafie warstwowym (def. 7)
 v – wierzchołek grafu (def. 15)
 val – częściowa funkcja przypisująca wartości danego atrybutu wierzchołkom (def. 3)
 x – wierzchołek lub krawędź (def. 3)
 x – warstwa (def. 4 i kolejne)

Dodatek B – Słowniczek pojęć

Definicje pojęć dotyczących gier użytych w pracy. Po znaku: // struktura odpowiadająca pojęciu w modelu grafowym.

Fabula – model mentalny historii wyrażający się w zdarzeniach definiujących świat i postaci oraz zależności między nimi, sens i cel zdarzeń. Na jego podstawie tworzona jest narracja.

Narracja – zbiór wszystkich obiektów systemu oraz reguł łączących te obiekty i określających możliwości ich modyfikacji i relacje między nimi (wydarzenia i związki przyczynowo-skutkowe) w sposób niesprzeczny z fabułą. Jest podstawą do zdefiniowania mechanik, reguł growych oraz świadomości fabularnej postaci (zbioru informacji fabularnych dostępnych dla postaci).

Gra (ang. *game*) – system, w którym gracz wykonuje działania w ramach świata gry, aby osiągnąć cel. Reguły, według których działa, i cele do osiągnięcia określone są w fabule i mechanikach gry. // GWSG i zbiór produkcji wraz ze zdefiniowanymi warunkami wygranej i porażki.

Gracz (ang. *player*) – użytkownik systemu, inicjujący działania jednej postaci równolegle (w grze czasu rzeczywistego) lub naprzemiennie (w grze turowej) z działaniami innych postaci. // Wskazany węzeł GWSG z warstwy postaci.

Gra fabularna (ang. *adventure game, role-playing game*) – gra, w której gracz kontroluje bohatera poruszającego się po świecie gry, wchodzącego w interakcje z innymi postaciami i dąży do osiągnięcia celu zdefiniowanego jako konsekwencja wydarzeń zachodzących w świecie. Związki przyczynowo-skutkowe między różnymi wydarzeniami i relacje pomiędzy obiektami w grze zdefiniowane są na podstawie fabuły. Struktura narracyjna uporządkowana jest poprzez misje, choć możliwe jest wykonywanie działań wychodzących poza ramy aktualnej misji.

Świat gry (ang. *game world*) – zbiór obiektów zdefiniowanych w grze i łączące je zależności. // GWSG.

Gra o zamkniętym świecie (ang. *close world game*) – gra, w której dla danego stanu świata gry dostępna jest jedna lub kilka produkcji węzłowych. Sekwencja rozgrywki dla takiej gry z dokładnością do produkcji węzłowych jest podciągiem jedynej lub jednej z kilku sekwencji zakończonych wygraną.

Gra o otwartym świecie (ang. *open world game*) – gra, w której dla danego stanu świata gry dostępne jest wiele produkcji węzłowych. Jest wiele różnych z dokładnością do produkcji węzłowych sekwencji zakończonych wygraną.

Lokacja (ang. *location*) – przestrzeń w świecie gry, w ramach której możliwe są interakcje między obiektami znajdującymi się w tej przestrzeni. Lokacje są połączone ze sobą, co oznacza możliwość przejścia postaci z jednej lokacji do drugiej. // Węzeł GWSG z warstwy „Locations”. Krawędzie pomiędzy tym węzłem a węzłami z innych warstw oznaczają obecność obiektów w lokacji. Obecność obiektów w jednej lokacji umożliwia interakcję między nimi. Węzły lokacji połączone są relacją dostępności.

Postać (bohater) (ang. *character*) – obiekt w grze aktywnie wpływający na jej przebieg. W szczególnych przypadkach może być pozbawiona możliwości ruchu, czyli wpływania na świat gry. // Węzeł GWSG z warstwy „Characters”.

Przedmiot (ang. *item*) – obiekt w grze mogący być elementem produkcji, nie mogący jej inicjować. // Węzeł GWSG z warstwy „Items”.

Informacja fabularna – stwierdzenie o świecie gry, które jest dostępne dla danej postaci lub związane z danym przedmiotem. Może umożliwiać wykonanie jakiejś produkcji lub skłaniać gracza do zainicjowania jakiejś produkcji. // Węzeł GWSG z warstwy „Narration”.

Misja (ang. *quest*) – ciąg działań gracza, mających określony cel (określone cele), którego osiągnięcie umożliwia wykonanie kolejnych misji lub wygraną. // Sekwencje produkcji najczęściej rozpoczęte produkcją wprowadzającą misję a zakończone jedną z produkcji końcowych danej misji.

Wątek fabularny (ang. *story arc, narrative arc*) – ciąg wydarzeń będących w związku przyczynowo-skutkowym. // Sekwencja produkcji, w której elementy modyfikowane w jednej produkcji zawierają się w lewej stronie kolejnej produkcji.

Produkcja końcowa misji/gry – produkcja, której efektem jest osiągnięcie warunku wygranej misji/gry.

Wygrana (ang. *win*) – ukończenie gry po osiągnięciu celu.

Porażka (ang. *fail*) – ukończenie gry wskutek braku możliwości wykonania jakiegokolwiek istotnego działania (w szczególności stan po usunięciu postaci sterowanej przez gracza ze świata gry (śmierci postaci)).

Warunek wygranej (ang. *win condition*) – podgraf świata gry zdefiniowany jako osiągnięcie celu misji/gry.

Warunek porażki (ang. *fail condition*) – podgraf świata gry uniemożliwiający zastosowanie jakiegokolwiek produkcji węzłowej. // Najczęściej jest to węzeł z atrybutem: **IsDead** odpowiadający postaci sterowanej przez gracza.

Działanie (akcja) – pojedyncza czynność wykonana przez bohatera zmieniająca świat gry. // Pojedyncza produkcja.

Wydarzenie – zmiana stanu świata gry na skutek działania gracza lub innej postaci.

Działanie węzłowe – działanie przybliżające gracza do osiągnięcia celu (wygranej). // Produkcja, w wyniku zastosowania której zwiększa się liczba istotnych możliwych do wykonania akcji.

Bohater niezależny (NPC, ang. *non-playable character*) – w grach fabularnych każda postać, w którą nie wciela się żaden z graczy.

Mechanika gry (ang. *game mechanics*) – reguła wprowadzająca możliwości i ograniczenia działań gracza. Z mechanik wynikają produkcje generyczne. Mechaniki określają także zasady działania systemu nieokreślone w modelu formalnym (np. wybór produkcji ze zbioru o identycznej lewej stronie) i są podstawą do implementacji.

Gra turowa (ang. *turn-based game*) – gra, w której świat gry zmieniany jest kolejno przez postaci w grze działające w zdefiniowanej kolejności.

Gra czasu rzeczywistego (ang. *real-time game*) – gra, w której świat gry zmieniany jest równolegle przez postaci.

Ekwipunek (ang. *inventory*) – zbiór przedmiotów będących własnością postaci. // Zbiór węzłów z warstwy przedmiotów połączonych relacją posiadania z jedną postacią.

Rozgrywka (ang. *gameplay*) – sekwencja zdarzeń w grze od stanu początkowego do wygranej albo porażki. // Sekwencja produkcji wybieranych przez gracza lub następujących pomiędzy ruchami gracza (w grze turowej) od stanu początkowego do stanu końcowego (wygranej albo porażki).

Dodatek C – JSON Schema

```
{
  "$schema": "http://json-schema.org/draft-07/schema",
  "$id": "http://www.ztg.fais.uj.edu.pl/",
  "title": "StoryGraph",
  "description": "JSON Schema for productions.",
  "type": "array",
  "definitions": {
    "node": {
      "type": "object",
      "anyOf": [
        {"required": ["Name"]},
        {"required": ["Id"]}
      ],
      "properties": {
        "Name": {"type": "string"},
        "Id": {"type": "string"},
        "Comment": {"type": "string"},
        "IsObject": {"type": "boolean"},
        "Characters": {
          "type": "array",
          "items": {
            "$ref": "#/definitions/node"
          }
        }
      },
      "Items": {
        "type": "array",
        "items": {
          "$ref": "#/definitions/node"
        }
      },
      "Narration": {
        "type": "array",
        "items": {
          "$ref": "#/definitions/node"
        }
      }
    }
  }
}
```

```

    },
    "Connections": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "Destination": {"type": "string"},
          "Attributes": {
            "type": "object",
            "patternProperties": {
              "^.*$": {
                "anyOf": [
                  {"type": "string"},
                  {"type": "boolean"},
                  {"type": "number"},
                  {"type": "null"}
                ]
              }
            }
          },
          "additionalProperties": false
        }
      },
      "additionalProperties": false,
      "required": ["Destination"]
    }
  },
  "Attributes": {
    "type": "object",
    "patternProperties": {
      "^.*$": {
        "anyOf": [
          {"type": "string"},
          {"type": "boolean"},
          {"type": "number"},
          {"type": "null"}
        ]
      }
    }
  },

```

```

        "additionalProperties": false
    },
    },
    "additionalProperties": false
}
},
"items": {
    "type": "object",
    "properties": {
        "Title": {"type": "string"},
        "TitleGeneric": {"type": "string"},
        "Description": {"type": "string"},
        "Override": {"type": "integer", "enum": [0, 1, 2]},
        "Comment": {"type": "string"},
        "LSide": {
            "type": "object",
            "properties": {
                "Locations": {
                    "type": "array",
                    "items": {
                        "anyOf": [
                            {"$ref": "#/definitions/node"}
                        ]
                    }
                }
            }
        },
        "additionalProperties": false,
        "required": ["Locations"]
    },
    "RSide": {
        "type": "object",
        "properties": {},
        "additionalProperties": false
    },
    "Preconditions": {
        "type": "array",
        "items": {
            "anyOf": [

```

```

        {
            "type": "object",
            "properties": {
                "Cond": {"type": "string"}
            },
            "additionalProperties": false,
            "required": ["Cond"]
        },
        {
            "type": "object",
            "required": ["Count"],
            "anyOf": [
                {"required": ["Min"]},
                {"required": ["Max"]},
                {"required": ["Min", "Max"]}
            ],
            "properties": {
                "Count": {"type": "string"},
                "Min": {"type": "integer"},
                "Max": {"type": "integer"}
            },
            "additionalProperties": false
        }
    ]
},
"Instructions": {
    "type": "array",
    "items": {
        "type": "object",
        "properties": {
            "Op": {"type": "string", "enum": ["create", "delete", "move",
"set", "unset", "add", "mul", "winning", "copy"]},
            "In": {"type": "string"},
            "To": {"type": "string"},
            "Attribute": {"type": "string"},
            "Expr": {"type": "string"},
            "Value": {

```

```

        "anyOf": [
            {"type": "string"},
            {"type": "boolean"},
            {"type": "number"},
            {"type": "null"}
        ]
    },
    "CharactersLimiter": {"type": "string", "enum": ["prohibit",
"move", "delete"]},
    "ItemsLimiter": {"type": "string", "enum": ["prohibit", "move",
"delete"]},
    "ChildrenLimiter": {"type": "string", "enum": ["prohibit", "move",
"delete"]},
    "NarrationLimiter": {"type": "string", "enum": ["prohibit",
"move", "delete"]},
    "Limit": {"type": "integer"},
    "Count": {"type": "integer"},
    "Sheaf": {"type": "object"}
},
"allOf": [
    {
        "if": {"properties": {"Op": { "enum": ["set"] }}}},
        "then": {
            "required": ["Attribute"],
            "anyOf": [
                {"required": ["Value"]},
                {"required": ["Expr"]}
            ]
        }
    },
    {
        "if": {"properties": {"Op": { "enum": ["add"] }}}},
        "then": { "required": ["Attribute", "Value"] }
    },
    {
        "if": {"properties": {"Op": { "enum": ["mul"] }}}},
        "then": { "required": ["Attribute", "Value"] }
    },

```

```

{
  "if": {"properties": {"Op": { "enum": ["delete"] }}}},
  "then": {
    "anyOf": [
      {"required": ["Node"]},
      {"required": ["Nodes"]}
    ],
    "properties": {
      "Node": {"type": "string"},
      "Nodes": {"type": "string"}
    }
  }
},
{
  "if": {"properties": {"Op": { "enum": ["create"] }}}},
  "then": {
    "required": ["In"],
    "anyOf": [
      {"required": ["Node"]},
      {"required": ["Sheaf"]}
    ],
    "properties": {
      "Node": {"$ref": "#/definitions/node"}
    }
  }
},
{
  "if": {"properties": {"Op": { "enum": ["move"] }}}},
  "then": {
    "required": ["To"],
    "anyOf": [
      {"required": ["Node"]},
      {"required": ["Nodes"]}
    ],
    "properties": {
      "Node": {"type": "string"},
      "Nodes": {"type": "string"}
    }
  }
}

```

```
        }
    }
]
},
"additionalProperties": false,
"required": ["Op"]
}
},
"additionalProperties": false,
"required": ["Title", "TitleGeneric", "Description", "LSide", "RSide",
"Instructions", "Override"]
}
}
```

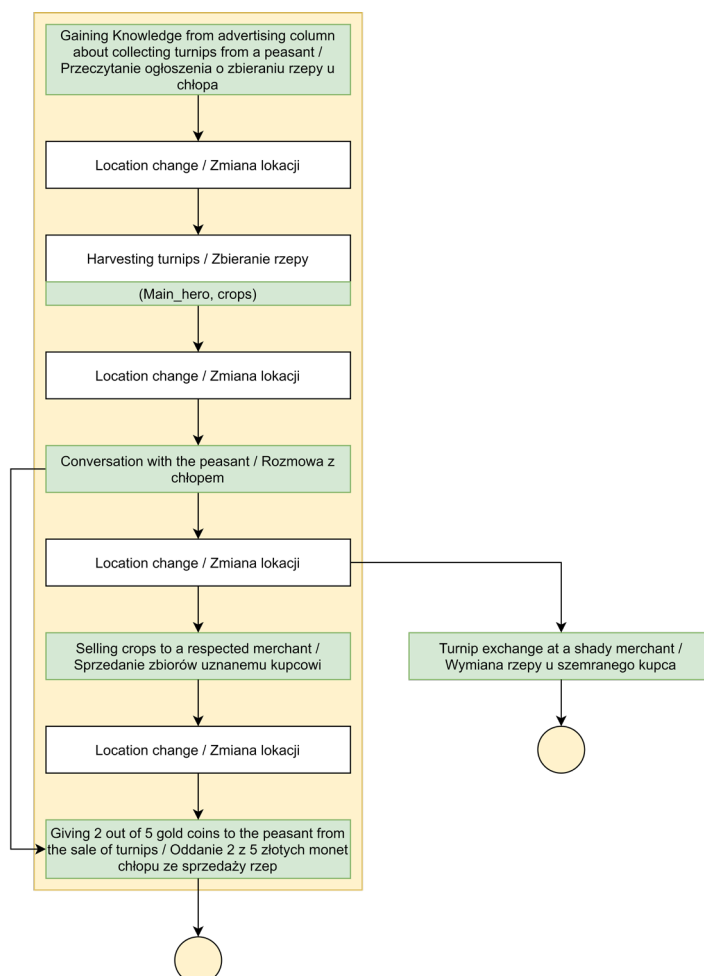
Dodatek D – Wybrane misje gry fabularnej stworzone przez studentów

Przykłady trzech misji stworzonych przez studentów w ramach zajęć z „Projektowania wspomagane go komputerem”. W załączniku znajduje się jedna misja początkowa, jedna środkowa i jedna końcowa. W diagramie misji środkowej i końcowej można zauważyć bloki wskazujące elementy narracyjne będące konsekwencją wypełnienia poprzednich misji (kolor fioletowy).

Misja „Help in the field”

Misja początkowa. Bohater przechadzając się po wiosce zwraca uwagę na słup ogłoszeniowy. Znajduje się tam ogłoszenie chłopca, który potrzebuje pomocy przy pracach w polu w zamian za złote monety. Bohater najmuje się do zbioru rzepy. Wraca do chłopca, który mówi mu, że ostatnie pieniądze niestety wydał, ale będzie w stanie opłacić zlecenie, jeśli bohater sprzeda zbiory na targowisku. Bohater udaje się na targowisko, gdzie ma do wyboru sprzedać rzepę, wrócić do chłopca i oddać mu część monet lub wymienić rzepę na bezcenny argentyński kaktus u szemranego kupca, wtedy chłop czuje się oszukany, a za bohaterem wystawiony jest list gończy.

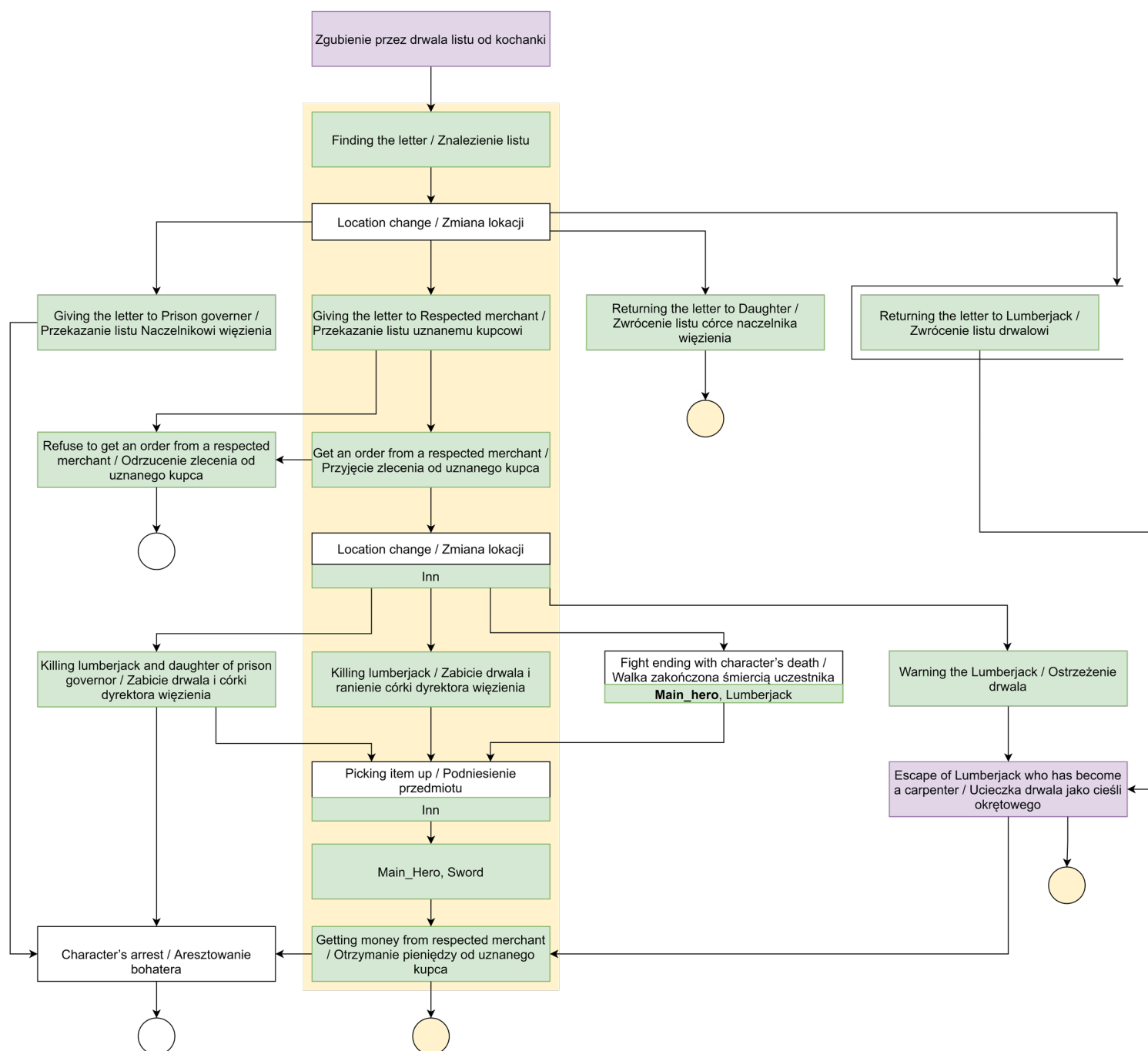
Autorzy: Paweł Łosek, Szymon Piechaczek, Filip Błaszczak



Misja „Hacking in Inn”

Misja środkowa. Bohater znajduje list żony kupca i córki naczelnika więzienia do jej kochanka. Wietrzy interes. Może zwrócić się z propozycją oddania listu w zamian za brzęczącą monetę do autorki, do jej męża, ojca lub do adresata. Każde z nich inaczej zareaguje na próbę szantażu. Autorka zapłaci bez szemrania, jej ojciec skorzysta z władzy, żeby się pozbyć listu i pechowego szantażysty za jednym zamachem, mąż zapala żądzą zemsty a adresat postanowi uciec.

Autorzy: Adamski Kamil, Szarek Filip, Żelichowski Filip



Misja „Taking over the ship”

Misja końcowa. Czarny czarodziej przy pomocy piratów przejął statek (załoga z kapitanem jest przetrzymana na statku pod pokładem) i go kontroluje. Statek, jest jedyną możliwą drogą do dostania się na wyspę, którą chce przejąć pod swoje władanie wraz z piratami, lecz żeby wypłynąć potrzebuje on kompasu. Gracz ma do wyboru: oferuje mu kompas (jeśli go nie ma, wybiera się na poszukiwanie) i dociera do zamorskiej krainy jako pomocnik czarnego charakteru, lub podejmuje walkę z czarodziejem, aby przekazać statek prawowitej załodze. To ostatnie uda mu się tylko, jeśli dysponuje magicznym artefaktem zdobytym w jednej z poprzednich misji.

Autorzy: Dawid Paluch, Maciej Mularski, Daniel Kułaga

